HYBRID PROCESSING

Thesis by

Christopher R. Carroll

In Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

Technical Report 5034

California Institute of Technology

Pasadena, California

1982

(Submitted March 8, 1982)

ACKNOWLEDGMENTS

The research described here benefitted me not only by the amount of knowledge I gained but also by the number of top quality people with whom I have had a chance to work. I want to thank each of them for sharing their thoughts with me.

In particular, I thank Dr. Ivan Sutherland for his valuable insights which got me off the ground on this project, and Dr. Carver Mead for his technical guidance which made it a success. I also want to thank the people at Xerox PARC and at the Information Sciences Institute for their technical help in fabricating my chips. I gratefully acknowledge financial support from Caltech, the National Science Foundation, and the Office of Naval Research, which supported this work in part.

Finally, I want to thank my parents for encouraging me toward this goal, and, most importantly, my wife Sylvia, without whom I could never have finished.

ABSTRACT

The past decade has witnessed a revolution in digital electronics. As the cost per function has decreased, digital techniques have pushed the older analog methods into the background. This thesis explores a method of merging digital and analog techniques into a hybrid combination of the two. Representing the analog information as continuously variable intervals of time minimizes the effects of noise on the analog data. Ensuring that only digital data pass from one computation to another prevents the accumulation of errors.

As an example of hybrid processing, this thesis includes the design of a Large Scale Integrated (LSI) circuit that implements the Lee-Moore maze solving algorithm, extended to cover the two-layer path finding case. The use of digital information to describe the path geometry and analog information to describe the path costs demonstrates the system's hybrid nature.

The design of this system provided several lessons applicable to the design of other hybrid systems. It also unexpectedly demonstrated the importance of the communication structure in determining the costs involved in all kinds of processing. These lessons are summarized in the last chapter.

TABLE OF CONTENTS

Chapter 1

WHAT IS HYBRID PROCESSING?

Before beginning to study hybrid processing, one must first know what the term means. The purpose of this chapter, then, is to provide the required definition.

Taking the definition a piece at a time, I must first convey my understanding of processing in general. In the context of this paper, "processing" refers to any operation performed on data that has some effect on those data. Processing operations tend to fall into one of two catagories. These are computation and communication.

On the computation side of processing, the operation performed on the data consists of transforming the values of those data according to some specified function. This is the part of processing that actually does the

arithmetic and other calculations that are responsible for "finding the answer" to a problem.

On the communication side of processing, the operation performed on the data consists of moving the information from one place to another. In an electronic processing element, this generally involves one or more wires over which the electrical signals representing the data travel. Without communication elements to deliver data to the inputs of a computation element and to carry away its results, processing machines could accomplish next to nothing.

Today, the importance of the communication structure within a processor is finally coming to light. Computation is now cheap, and can occur simultaneously in a very large number of elements within a system. The issue of how to communicate the data from one computation element to another at the right time and in the right sequence dominates many modern processor designs. Communication and computation both play important roles in the structure of any processing strategy today.

Beyond the computation versus communication classification, processing elements can be classified

further by the type of data on which they operate. Again, there are two recognizably separate catagories to consider. These are known as analog processing and digital processing.

Analog processing deals with data that are continuously variable. A slide rule is a familiar example of an analog processor. The position of the slide forms the input to the computation, and the reading on the scale under the cursor gives the output. The slide is continuously adjustable. There are no notches, or steps in its movement. Similarly, the scale on which the output is obtained under the cursor is a continuous scale. The precision available in the output, or "answer", of such a device is theoretically unlimited. Since there are no indivisible units involved, variables can take on exact values. In practice, the available precision is limited by the accuracy with which the machine is built. In the case of the slide rule, the accuracy with which the scales are marked on the instrument is the limiting factor. To summarize, then, analog processors work on continuous data with unlimited precision, but with accuracy limited by the accuracy of construction of the machine.

Digital processing deals with data that are quantized. In any such device, there is an indivisible unit of information, the smallest value above zero that the machine recognizes. An abacus is a familiar example of a digital processor. Data are entered by discrete movement of the beads. Each bead is against either one stop or the other, never somewhere in between. When one reads the answer from an abacus, each bead is counted if it is on one side of the device, or not counted if it is on the other side. The accuracy of this kind of processor is unquestioned. After all, there is no problem in determining whether a bead should be counted in the output or not. The precision available, however, is limited by the size of the machine, or, in the case of the abacus, the number of rows of beads in the machine. To summarize, digital processors work on discretized data with unlimited accuracy, but with precision limited by the construction of the machine.

With these definitions of digital and analog processing in mind, one can go off happily classifying almost any kind of processing equipment around. Networks of resistors provide an example of analog arithmetic, where the voltage on one node of the network is a linear function of the other node voltages. A mechanical device

known as a planimeter performs analog integration in finding the area enclosed by plane curves. Common digital processing devices include things as small as digital watches and pocket calculators as well as the largest mainframe electronic computers around today.

A hybrid processor is nothing more than a processor that uses a mixture of both digital and analog techniques. As usual, when there is a choice of methods to use in attacking a problem, there are circumstances under which each of the possible choices is the best. Thus in building a processing machine, a designer, by taking note of the circumstances surrounding his design, can choose to employ either digital or analog techniques, as appropriate. Under some special circumstances, a combination, or hybrid, of the two is the proper choice to take.

The fact that digital and analog processing work on different kinds of data means that they have some different characteristics that might lead the designer to prefer one method over the other in his design. The next two chapters will investigate some of those differing characteristics, and disclose those circumstances when hybrid processing might be advantageous.

Chapter 2

WHY USE HYBRID PROCESSING?

Many of today's designers of processing equipment are nervous about the idea of using any analog techniques at all in their machines. They argue that purely digital techniques can solve any processing task, and can eliminate the main problem that plagues analog designs, noise.

Noise is any unwanted signal from the environment of a system that perturbs the data in that system in some way. Digital and analog structures differ in their response to noise, and this difference is a major consideration when choosing a design strategy.

Analog processors, like the slide rule, have no immunity to noise at all. On a slide rule, noise could

have the effect of causing the slide or the cursor to slip slightly away from the position meant for it. This would introduce an error in the computations that is indistinguishable from real data. Thus, in analog processors, noise gets carried along through the rest of the computations and can adversely affect the result.

Digital processors, like the abacus, can tolerate some noise. On the abacus, noise could cause the beads to stray slightly away from their ideal positions at one end or the other of the rows. However, as long as the beads remain closer to the side where they belong than to the other side, their correct values are still readable. Unless the noise is so severe that it causes the beads to travel more than halfway to the other side of the abacus, it does not affect succeeding calculations in any way. Digital processors have some inherent immunity to noise because after each computation, digital variables are restored to a clean representation of one of the discrete values that the data can represent. Any errors generated by noise are eliminated at each step of the processing, and thus cannot accumulate.

It is usually easy to distinguish analog and digital problems from each other. Analog problems are those that

arise from the measurement of some quantity. In general, the variables tell how much of something one has, in a qualitative way. Digital problems, in contrast, arise from counting situations. Digital variables tell how many somethings one has. As examples of problems at the opposite ends of the analog-digital spectrum, consider the two mundane problems of balancing a checkbook and controlling water temperature in a shower.

Think about the problem of balancing a checkbook. Given the choice of the two processing machines shown in Figure 2-1, which is the most appropriate one with which to attack the problem? Each of the machines will compute sums from zero to $1000.00. On the abacus, the additions and subtractions involved are performed by discrete movement of the beads in the rows. On the other instrument, similar to a slide rule, the slide is positioned with the zero mark on the present balance in the checking account, and the cursor is then moved to lie over the amount of the deposit or withdrawal on the slide's scale. The resulting new balance is read under the cursor from the scale on the body of the instrument.

Now, consider the problem of designing an automatic water temperature control system for a shower. Which of

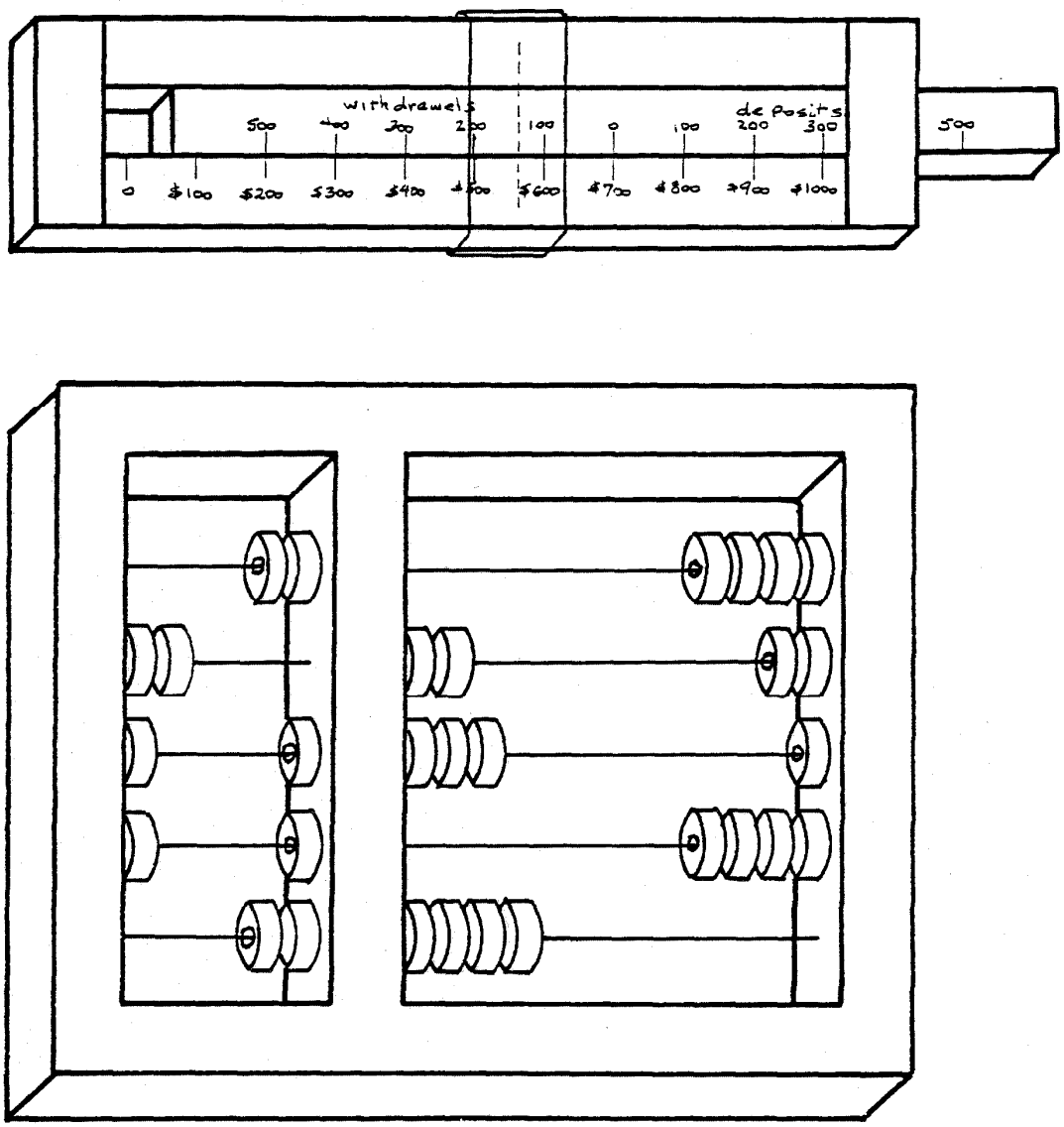
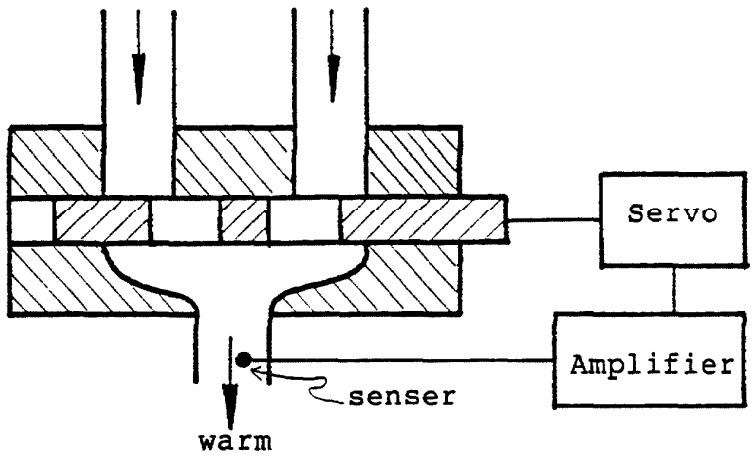


Figure 2-1. Which machine is better for balancing a checkbook?

the two approaches shown in Figure 2-2 is most appropriate for attacking this problem? In one case, the mix of hot and cold water is continuously adjustable by means of the position of the diaphragm in the mixing valve. In the other case, the mix is determined by the ratio of the number of hot water valves that are on to the number of cold water valves that are on. In this second case, the adjustment in the mix is not continuous, but adding more and more valves in the system can result in enough precision in temperature control to satisfy any set requirements.

For the checkbook problem, the better choice of instrument is clearly the abacus. The slide rule device has two major deficiencies that would preclude its use. The first concerns number representation. To represent a number as large as $1000.00 accurately to the last cent would require an accuracy of one part in 100,000 in the marking of the scales on the instrument, which is unrealistic. The second deficiency of the slide rule instrument in this example comes from noise problems. With each transaction computed on the checkbook balance, some noise would unavoidably enter into the calculations. After only a few transactions, the noise-introduced errors would accumulate enough to swamp out the pennies and dimes



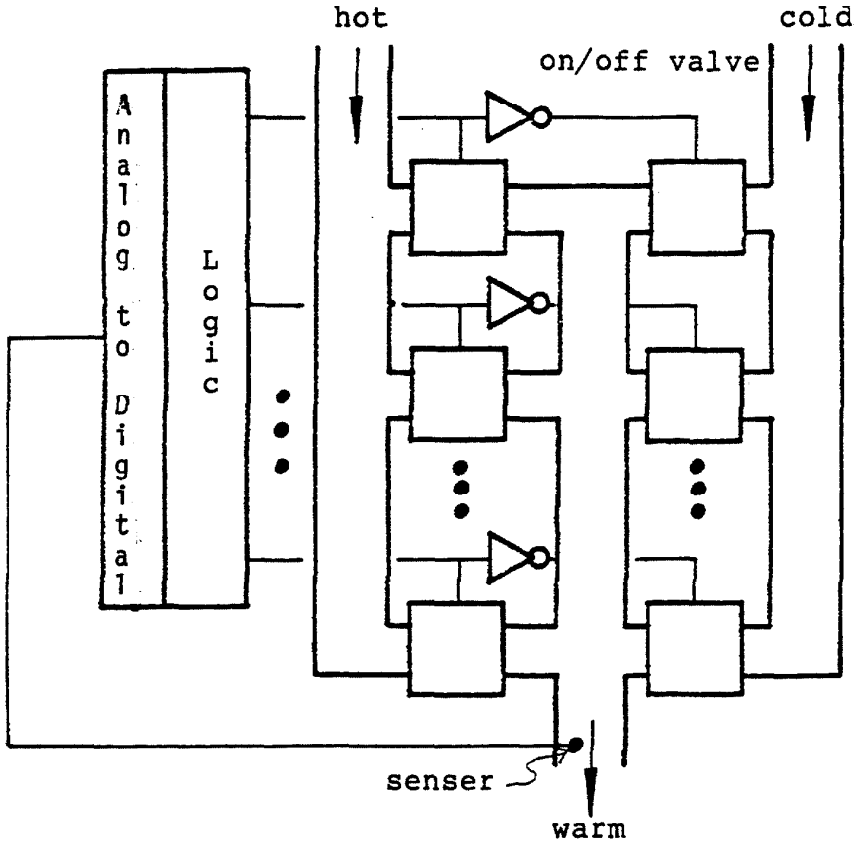


Figure 2-2. Which system is better for controlling water temperature?

in the calculations, and the resulting balance in the checkbook would no longer agree with the amount recorded in the bank. The abacus, on the other hand, can easily represent the variables precisely and without danger of errors introduced by noise. Since the data are restored to clean, noise-free values after each transaction, the noise-generated errors have no chance to build up.

The choice of approach to the water temperature control system problem may not be as apparent. Although the problem clearly involves continuous variables, and is thus analog in nature, either approach can lead to a design with good performance. The analog approach, with the single mixing valve, can deliver a continuously variable mix of hot and cold water, which is desirable. The other, digital, approach can adjust the mix only in discrete steps, but by using enough valves, the bather can be completely unaware of the discrete changes in water mix. What, then, is another basis for choosing between the two schemes? One significant difference lies in the complexity on the communication side of the processing. The digital solution requires a separate water pipe and a separate solenoid control signal for each of the many valves in the system. The single valve in the analog design requires a considerably simpler, and thus less

expensive, communication arrangement, both electrically and mechanically. Overall, then, the analog solution to this fundamentally analog problem is a better choice than the digital solution, even though both can adaquately perform the job.

In today's world, I would be tempted to guess that, although most readers of this paper correctly solved the checkbook problem above, some readers chose the digital solution to the shower temperature control problem despite its disadvantage in complexity. Designers today have almost unlimited amounts of hardware with which to build structures, and digital solutions are easy to most problems. As a result, digital techniques pervade the thoughts of designers, and force analog techniques down into the cellars of their minds. The shower temperature control problem should have demonstrated that there still is a need for analog treatment of some situations.

So today, digital processing is cheap, yet there is still a need to input parameters or generate outputs that are continuous variables, best suited to analog techniques. This is the setting where hybrid processing shines. With hybrid processing, the best of both worlds is available. Ideally, such a processor could handle

discrete data with digital circuitry and continuous data with analog circuitry. The trick is to merge the two types of processing so that the variables, discrete and continuous, can interact to solve the problem.

A strong tendency amoung current designers when trying to incorporate analog and digital variables in one design is to use analog-to-digital and digital-to-analog converters to isolate from each other the parts of the machine dealing with different kinds of variables. This results in a degenerate case of hybrid processing, where the machine is merely an interconnection of separate analog and digital processors. The strength of hybrid processing becomes clear only when the digital and analog parts of the system are merged so closely that they are not separately identifiable.

The game of backgammon illustrates a situation in which hybrid processing could find effective use. The position of the playing pieces and their movement according to the throw of the dice are purely digital quantities. However, the choice of which moves to make when there are several possibilities is based on a number of purely analog variables. One of these is the overall player strategy, offensive, defensive, or somewhere in

between. Another analog variable is the playing ability of the opponent, for some moves, though dangerous with a skilled opponent, may go unnoticed by an amateur. The computation involved in making the choice between moves must produce a digital answer, i.e. the one move that the process selects. The inputs to that computation are the digital quantities of board position and dice throw and the analog qualities which determine which decision is best.

The title of this chapter is "Why use hybrid processing?" The answer is that hybrid processing is the natural way to tackle many of today's problems. Many calculations that require lengthy or precise computation and therefore need digital data representation depend upon parameters that come from the real world, where quantities tend to be continuous. In such cases, hybrid processing can bring together the two forms of data to produce the desired result.

Chapter 3

WHEN TO USE HYBRID PROCESSING

When a designer has a processing problem to solve, what tests can he apply to decide between an analog implementation, a digital implementation, or some hybrid combination of the two? This chapter will present some criteria for making that judgment.

3.1 When to go Analog

Purely analog processing finds applications that surround our daily life. The world we live in is built around continuous variables and the differential equations that govern them.

Analog problems typically arise from measurements of quantities in the real world. The presence of variables

that describe how much of something, in a qualitative way, indicates that analog techniques may be applicable.

Analog computing methods offer an excellent way to make a qualitative analysis of a problem, but are limited in other applications. The accuracy of analog methods, as discussed before, is limited. Practical accuracies of one part in a thousand or so are the best that one can obtain. For higher accuracy, digital computation methods must come into play.

As shown in the example of balancing a checkbook in Chapter 2, the accuracy of a single computation is not the only limit to the usefulness of analog computation. Errors introduced by inaccuracy or noise tend to accumulate in analog systems. When a task demands that data pass through many stages of processing the use of analog techniques would eventually swamp out the data by the accumulating errors. In this case, a digital strategy is the only way to go.

Remembering the example of the shower temperature control problem, analog methods offer a saving in hardware complexity. Thus, when the cost of hardware is important, one should consider analog approaches to the problem.

In summary, the above results suggest some tests for deciding when to use analog processing. Analog processing is indicated when

- .the problem deals entirely with continuous variables
- .variables record qualitatively how much of something is present
- .the qualitative behavior of the problem is more important than the quantitative behavior, or
- .the hardware complexity must be minimized.

Analog processing is contraindicated when

- .any variable is discrete
- .high accuracy is important, or
- .data must pass through a large number of calculations or iterations.

By applying these tests to a particular design problem, a processor designer can determine if analog techniques hold any promise in that situation.

3.2 When to go Digital

In many cases, digital techniques provide the only acceptable solution to a design problem. The accuracy with which digital processors can perform calculations and

their inherent immunity to noise set them apart from the analog types of processors.

Problems whose variables count something, or tell how many somethings are present, are fundamentally digital problems. Balancing the checkbook in the preceding chapter was such a problem.

Some problems can be solved only by iterating through a sequence of calculations. Such iterations can result in millions or even billions of operations being performed on a piece of information before the answer materializes. These problems cannot tolerate the errors induced by inaccuracies and noise in analog systems. Only digital approaches are feasible.

In summary, digital processing is indicated when

.variables record a count, or answer the question
"How many?"
.the problem demands high accuracy
.data must pass through a large number of operations
to generate an answer, as in iteration, or
.the cost of hardware is not an issue.

Digital processing is contraindicated when an analog solution is possible and more cost effective. In such a case, the extra complexity of a digital solution merely adds extra baggage. If a problem passes the above tests, then digital methods demand some attention.

3.3 When to go Hybrid

Some problems fail to pass the tests for either purely analog or purely digital processing. These problems need some of the characteristics of each of the two approaches. It is for this class of problem that hybrid processing exists.

These problems tend to fall into two catagories. The difference lies in the reason for needing some digital processing.

The first subclass of hybrid problems consists of those tasks that require digital techniques merely to achieve adaquate accuracy. The variables involved in these problems are fundamentally analog quantities. They generally originate in some transducer or sensor and, possibly after a little analog processing, pass directly to an analog-to-digital converter. From that point on,

processing is totally digital. The results of the digital calculations sometimes are used as they are in digital form, but often they pass through some digital-to-analog converter back into analog form for display or for controlling some other analog device. This class of problems leads to the degenerate case of hybrid processing described earlier.

One finds examples of this kind of hybrid processing whenever data from the real world must be processed digitally. Digital voltmeters, for example, input an analog voltage, process it with an analog voltage divider to set the scale, and then digitize it for accurate measurement and display. A vector display system provides an example of a reverse situation, where the analog processing comes at the end of the process. Values representing the endpoints of the vectors are calculated digitally, but then converted to analog values. Analog processing elements can then calculate the deflection signals and intensity levels to send to the CRT for display [8].

The other subclass of hybrid processing involves problems that tend to be fundamentally digital, but that depend in some way on one or more qualitative, analog

inputs. The backgammon playing machine discussed earlier provides an example of this class of problem. The mechanics of play, like board position and the throw of the dice, are digital variables. The tactics of play, like strategy and skill of the opponent, are analog variables. On each play, the choice of moves to make is based on information from both kinds of sources.

It is this kind of hybrid processing that designers most ignore when building processing equipment. Instead of creating a structure that can effectively merge the analog and digital parts of the calculation, they provide some purely arbitrary digitization of the qualitative inputs and then build an entirely digital machine. Current game playing machines, for example, compete at one of several discrete levels of skill. The skill level, which should be an analog input, enters the calculations instead as an artificially digitized variable. The past twenty-five years of steadily decreasing digital hardware costs have trained designers to choose the digital route when they can find any possible way to do so. The result is that analog or hybrid solutions to problems are shelved in favor of a less direct digital solution. The general belief is that today's VLSI technology should continue that trend and push the digital revolution even farther

along, because of the enormous amount of hardware that can be realized on one chip of silicon. That, however, is a little misleading.

While the cost of computing hardware continues to fall, the communication hardware required to interconnect the computation elements has risen in relative cost. In VLSI technology in particular, as the computation elements continue to shrink, the proportion of chip area devoted to interconnection wires is increasing. The total cost of hardware, then, is not going to zero, but instead is becoming dominated by the cost of communicating data from one place to another.

As a result of this communication aspect of hardware cost, hybrid processing may begin to see more application. Transmitting only one bit per wire as is done most often in digital designs is not a very effective way to use the now costly communication path. Because of the fact that a VLSI chip is a very controlled environment in which noise levels are low and somewhat under the designer's control, many bits may be sent along one wire using analog techniques with a savings in cost right where it helps the most, in communication hardware.

Already some are discussing the possibility of multi-level signalling in digital circuits [9]. Microcode ROMs on some microprocessors store two bits per cell by using a four level encoding structure [10,11]. Although these designs are still strictly digital, because the allowed states for the data are still discrete, they are a step along the way toward full analog techniques. At least one chip designed at Caltech has made use of a ROM storing purely analog values in the design of a cursive character display generator [2].

Thus, there is motivation to encourage hybrid processing. Digital problems with qualitative, analog inputs exist, and the cost structure of current technology favors the hardware economy that analog techniques offer. The second half of this thesis will present a complete design that effectively makes use of hybrid processing. First, however, the next chapter will provide some guidelines on how analog and digital techniques might be merged to form a true hybrid processor.

Chapter 4

HOW TO USE HYBRID PROCESSING

Now that we know what hybrid processing is, why it is important, and when to use it, the question of how to implement it arises. It is not a trivial task to effectively merge analog and digital variables into a single hybrid processing structure.

Remember that true hybrid processing arises from problems that require primarily digital processing but that also depend in some qualitative way on one or more analog variables. The task, then, is to affect the results of a digital computation through the influence of analog variables.

What effect can an analog value have on digital data? The only way to modify digital values is in

discrete steps, changing bits from one state to another. By definition, there is no way to modify the value of digital data in a continuous way. Thus, on the surface, it would seem that the only way to get analog data into an otherwise digital computation is to first digitize it using some type of analog-to-digital conversion.

Information, however, need not always be represented by the value of some piece of data in a machine. The timing relationships between signals can also represent information in a problem.

Time offers some unique features as an analog variable. Time is a monotonic, smoothly increasing quantity. One can count on it to never reverse direction, or stop, or do anything but steadily continue forever. If one represents an analog value by the interval of time between two digital signals in a system, then that analog value will be immune to electrical noise to the same extent that digital values in that system are immune. Since time is measured in the same way in all parts of the system, and since small differences in time are much easier to detect than small differences in other analog values, representing analog data using intervals of time

results in better accuracy than would other choices of representation.

The human nervous system uses time to represent its data. Most people would agree that the body is a noisy environment in which to try to communicate information by minute electrical signals. Such an environment demands the use of a digital communication structure. However, all the data with which the body is concerned are in analog form. Light, sound, temperature, etc. all are analog inputs, and muscle and gland control signals are analog outputs of the nervous system. What a complicated mess it would be if all that data had to be converted from analog to digital form for communication on the nervous system! Instead, the body uses a much simpler strategy. The nerves carry impulses that are digital in nature, as they must be in order to overcome the noise. The frequency of the impulses, however, varies. The amount of time between impulses is the information carrying quantity [12]. That amount of time represents the analog value, and is continuously variable, so no conversion of the analog inputs and outputs of the nervous system is necessary. The analog signals are represented directly as varying intervals of time.

The combination of digital data processing with an analog dependence on a time variable offers an especially attractive opportunity for hybrid processing. If the analog dependency of a hybrid problem can be couched in terms of the timing of some digital function, then that timing can be the means for providing the required analog control over the calculations. Using this method requires no explicit conversion of the analog data to digital form. The interaction takes place as a result of varying the timing of the digital calculations.

The algorithm selected as a basis for a hybrid solution to a problem must allow the digital calculations to depend on some aspect of timing that can be modified by the problem's analog inputs. One way to achieve this dependence is to arrange for the digital logic to take one of several actions depending on which of several events happens first. The timing of the events can then be under the control of analog computations. This scheme successfully merges the analog variables into the digital computations in a natural way.

The physical configuration that is to implement a hybrid system also is constrained by the scheme of using time as an analog variable. In the first place, the

hardware cannot be fully synchronous. Synchronous systems digitize their time variable, so that no analog modulation of any timing function would be possible. Secondly, this approach works best when the problem maps well onto a processing structure that allows many operations to occur in parallel. Only with parallel processing can one use the technique mentioned above, using the difference in timing of several events to influence the digital calculations. In a structure with no parallelism, only one event can happen at a time, and no comparisons are possible.

The next four chapters will detail the development of a hybrid processor system that uses time to represent its analog data. It is not intended as an example of hybrid processing in general, but rather as a specific example of what hybrid techniques can accomplish. Chapter 5 will describe the problem to be solved by this system. Chapter 6 will develop some initial thrusts toward a solution, and finally, Chapters 7 and 8 will present the complete hybrid processing system and describe its operation in detail.

Chapter 5

AN EXAMPLE - - - LEE-MOORE PATH FINDING

This chapter begins the discussion of an application of hybrid processing in a particular system. The task that the resulting machine solves is the problem of path finding, which has application today in printed circuit board wire routing and interconnect wire routing on integrated circuits. This chapter will briefly outline the traditional Lee-Moore algorithm that is the basis for the path finding, and the following chapter will discuss an LSI implementation of the algorithm that solves the problem for paths on one layer. Succeeding chapters will examine the additional problems of two layer path finding, and will detail another LSI implementation that handles the two layer situation.

5.1 The Lee-Moore Algorithm

The Lee-Moore algorithm for path finding, proposed by Moore in 1959 [7] and extended by C. Y. Lee in 1961 [5], is a scheme for finding the shortest route between two points in a plane, where the route is composed of some number of vertical and horizontal segments through a rectangular grid superimposed on the plane. This has been a popular algorithm for people doing problems related to maze solving because it is easy to implement and because it guarantees that a path will be found if one exists. The drawbacks to the algorithm are that it is expensive computationally in both time and space. However, the use of the hardware to be described circumvents these difficulties.

Suppose that the size of the grid, i.e. the pitch of the cells defined by the grid, is set to the minimum path width that is allowed*. In the case of printed circuit board design, this would be the minimum center to center spacing for adjacent wires. Suppose also that the grid is uniform and symmetric, forming an array of square cells,

* Since our geometry here is based on this grid, the distances mentioned will be Manhattan distances, i.e. the distance from A to B would be the shortest distance covered in driving from A to B on the streets of Manhattan.

each a path width on a side. The path found by the algorithm from point A to point B will consist of a route beginning at the cell containing point A, continuing to a neighbor of that cell, and then to a neighbor of that second cell, and so on from a cell to one of its neighbors, until eventually the path ends in the cell containing point B. Some of the cells in the array may be blocked, preventing the path from running through these cells. These would be "barriers", or "walls" in a maze, or cells occupied by previously routed wires in the printed circuit board application.

The algorithm finds the shortest path from A to B in two phases. One word of storage, which I will call the "label", is associated with each of the cells in the array. The first phase, called the Propagation Phase, stores information in the labels throughout the array. The second phase, called the Retrace Phase, then uses that information to find the required path.

The Propagation Phase, which distributes the information, executes the following program:

```
put label -1 in all cells that are blocked
put label 0 in all cells that are not blocked
N:=1
put label N in the cell containing point A
while cell containing point B is labelled 0
   and more activity is possible do
begin
     for every neighbor of every cell labelled N do
          if that neighbor is labelled 0 then
               label it (N+1) else leave it alone.
     N:=N+1
end
```

This part of the algorithm is illustrated in Figure 5-1. The purpose of this phase is to distribute information to the cells that can then be used to find the direction back to point A. The information is spread out in a propagating wavefront centered on point A, much like waves propagating away from a stone dropped in a pond. It is interesting to note that the only activity that takes place occurs at the frontier of this expanding wavefront. Cells ahead of the frontier merely wait for the wave to arrive, keeping their label of 0. Cells behind the frontier have already received the information they need, and simply keep it stored in their label.

The Retrace Phase, using the information stored in the labels, executes the following program to find the path:

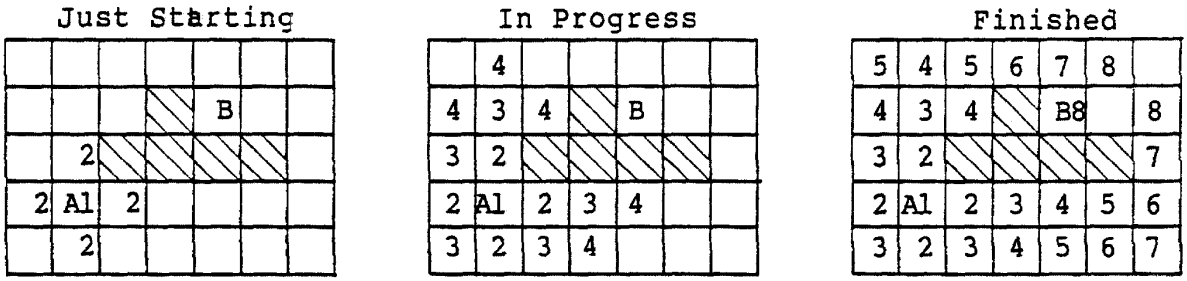


Figure 5-1. The Lee-Moore Propagation Phase

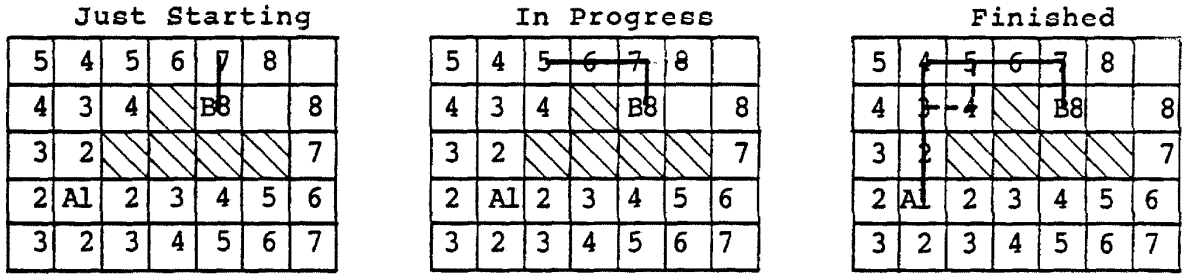


Figure 5-2. The Lee-Moore Retrace Phase

```
Start the path at the cell holding point B
N:= label of cell holding point B
if N=0 then there is no path from point A to point B
else begin
     while path has not reached cell holding point A do
     begin
          N:=N-1
          Continue the path to a neighbor of the
             current cell that contains the label N
     end
end
```

This part of the algorithm is illustrated in Figure 5-2. Notice that there is nothing to specify which cell to select when there are two or more possible choices. This merely means that there are multiple paths between A and B that have the same length. To first order, there is thus no preference of one path over another, so no selection mechanism need be used. In practice, some scheme is often employed when there is a choice of paths to take. A common selection scheme is to avoid changing directions in the path during the Retrace Phase when it is unnecessary. This tends to minimize the number of bends in the resulting path. When this phase is complete, the algorithm either has found the required path from A to B or has proven that no such path exists.

Before beginning the discussion of the hardware implementations of this algorithm, one should note a couple of things. First, examine the time complexity of the programs above. The Retrace Phase merely traces the

path from B back to A using information stored in the cells. The only cells accessed are those along the selected path and their immediate neighbors. The time complexity is thus linear with respect to the path length. However, in the Propagation Phase, the situation is worse. Information is propagated in all directions around point A. The number of cells accessed is approximately proportional to the square of the path length. Thus, the time complexity here is quadratic with respect to the path length, making the algorithm as a whole quadratic. This is unfortunate, since for maze solving to be interesting, a large maze must be involved. In the circuit board application, for example, a cell array containing 1000 x 1000 cells would be common. The quadratic time aspect of the algorithm thus is a real handicap. Current software using this algorithm to route typical printed circuit boards can consume several hours of CPU time on a full-size computer. On top of that, the space requirement is also large. Circuit board routing with this algorithm requires 10-12 bits of storage for each cell, and a million 12 bit words is a lot of memory. So, both the space and time complexity of the algorithm need to be attacked in any successful hardware implementation.

5.2 Hardware for Finding One Layer Paths

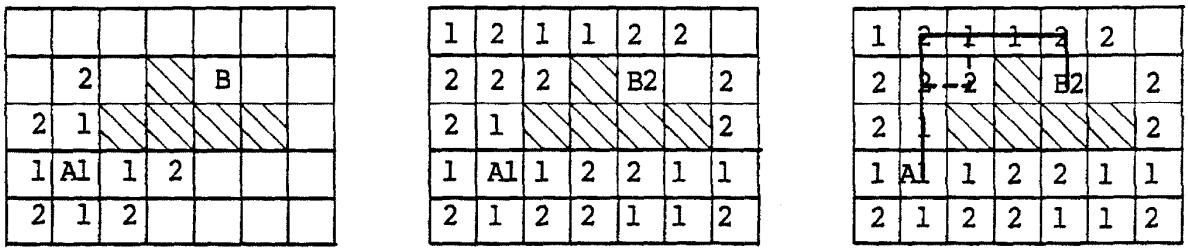
Implementing the Lee-Moore algorithm in hardware is conceptually a clean and natural thing to do. Because the problem is cellular and because information flows only between adjacent cells without using any long distance communication paths, the task is a natural one for an array processor structure with one processor per cell in the array. However, if there is to be any hope of building a large machine this way, there are two problems that must be overcome. First, the amount of storage per cell must be limited. In the original algorithm, in an array of unbounded size each cell would be required to contain an unbounded number of bits. Second, the global state required in the original algorithm, which was represented by "N" in the programs above, must be eliminated. Accomplishing these goals would result in a machine that could be extended to any size needed without undue complications.

The first goal, that of limiting the amount of storage required in each cell, was attacked by S. Akers in 1967 [1]. He showed that only two bits were required per cell to implement the algorithm proposed by Lee and Moore. Of the four states available from the two bits,

one indicated that a cell was blocked and unavailable for new paths. Another was used to indicate a cell that was so far untouched by the propagation process, like label 0 in the above programs. Then, instead of using the ascending ordinal numbers to label successive wavefronts in the propagation, Akers used successive members from the sequence

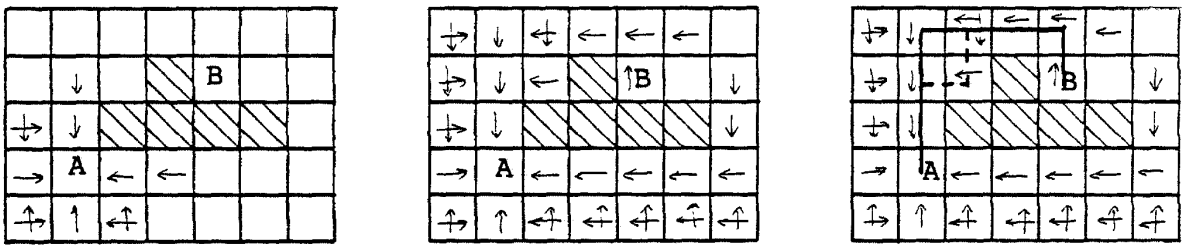
1,1,2,2,1,1,2,2,1,1,2,2,. . .

These last two states stored the information necessary to get back to the point where the propagation started. See Figure 5-3. It was only required that the program remember whether it was on the first 1, second 1, first 2, or second 2 in the sequence when it stored a number in the goal cell containing point B. For example, if the program knew it was on the second 1 of a pair when it reached the goal, then in the Retrace Phase it looked for cells containing the above sequence in reverse order, starting with the first 1, then 2,2,1,1, etc., until it reached the starting cell. This was a big step forward, for only two bits of storage were needed for each cell, no matter how big the array of cells was made. Unfortunately this did nothing to solve the problem of having to distribute the next member of the sequence as a global variable to all the cells in the array.



In Progress Propagation Complete Path Found

Figure 5-3. Akers' Modification



In Progress Propagation Complete Path Found

Figure 5-4. Modification for MAZER

The solution to the dilemma of global state is to use a slightly different strategy in the algorithm. Rather than numbering successive wavefronts with some sequence and then searching for the reverse of that sequence to find the path, simply store in each cell arrows that point to the neighbor(s) from which the wavefront approaches as it passes over that cell, and then just let the arrows show the path back to the starting cell. This approach is shown in Figure 5-4. Since the wavefront may reach a cell from more than one neighbor simultaneously, and since that fact is important when trying to select one of several equally good paths, an arrow for each neighbor is needed. These arrows require one bit each, because the wavefront either came from that neighbor or it didn't. Additionally, one bit is required to identify a cell as being blocked. Five bits per cell is more than Akers' two, but it is still a small number, and more importantly, it is still a bounded number, that does not change as the array of cells grows. With these modifications to the algorithm, the move to hardware is hardly more than just "wiring it up".

Chapter 6

THE LEE-MOORE ALGORITHM IN HARDWARE

As discussed in the preceding chapter, implementing the Lee-Moore path finding algorithm in hardware is a clean and natural thing to do. A machine consisting of an array of processors, with one processor per cell in the algorithm, matches the problem perfectly. This chapter will detail the design of such a hardware implementation, culminating in an LSI chip that performs Lee-Moore path finding.

6.1 A Demonstration Circuit

As a demonstration of the feasibility of this approach to path finding, I built a small array of processors out of standard TTL parts. Figure 6-1 shows the circuit I used. As can be seen from the figure, there is not much to the "processor". It consists of one and

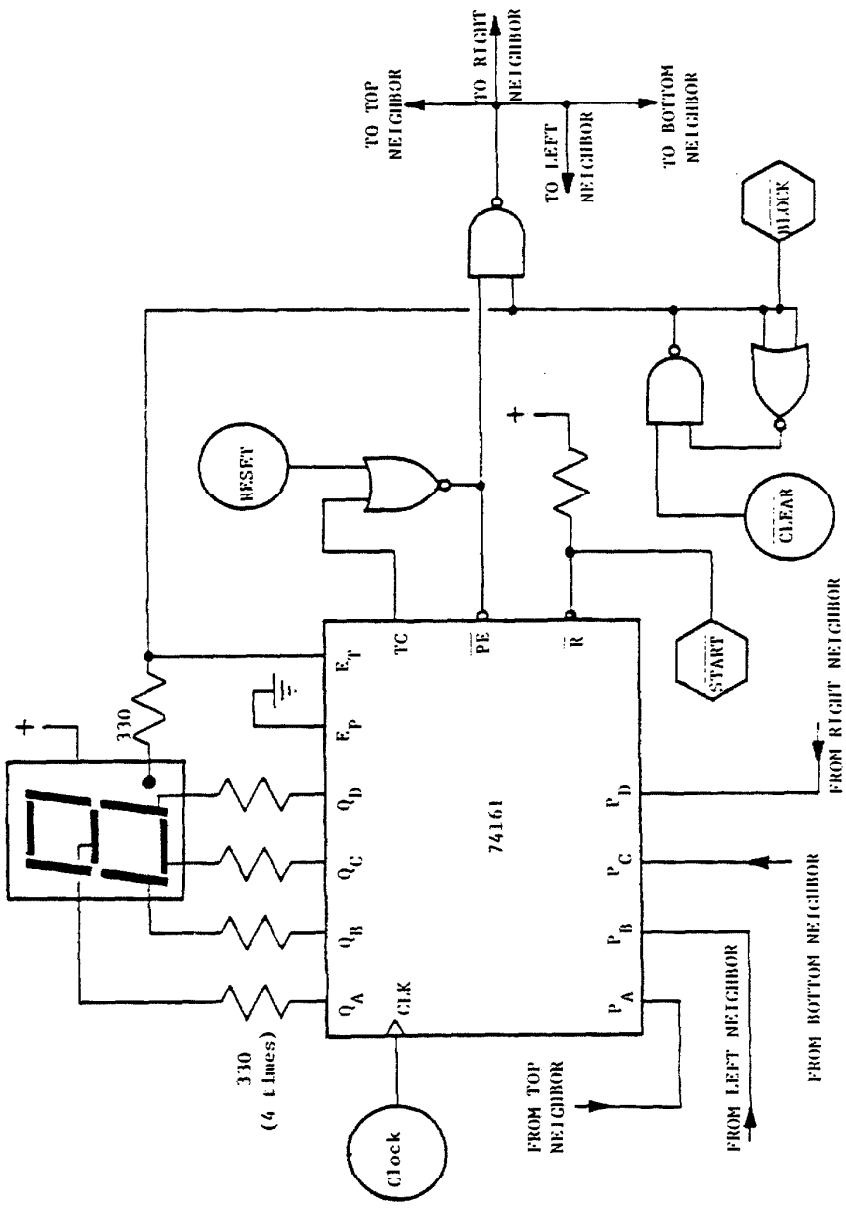


Figure 6-1. Schematic for Demonstration Circuit

two halves standard TTL packages, a few resistors, and an LED display. The circuit uses a 74161 as a four bit latch. The 74161 features the TC output, which is the logical AND of the four latch outputs and the ET input. The four bits in the 74161 are the four arrow bits. The fifth bit is formed by one NAND and one NOR gate to indicate the blocked condition. Global control signals are circled. The two signals START and BLOCK are independent for each cell and are activated by momentarily grounding that node with a probe tip. Incoming communication from neighbor processors enters this processor at the preset inputs of the 74161. Outgoing communication to the neighbors exits from the NAND gate at the right.

In operation, the circuit is quite simple. Initially, the CLEAR signal is taken low to clear all the block flip-flops. Then the maze walls are defined by selectively blocking some processors by grounding their BLOCK inputs. The LED decimal point lights in the blocked cells. Next, RESET is taken high for at least one clock cycle. This forces all communication wires between neighbor processors high and parallel loads all ones into the 74161, turning off the LED segments. This is a stable configuration, and will not change as the clock ticks.

All communication wires stay high, and the latches keep parallel loading all ones because the TC outputs are high. Now suppose that somehow the processor to the right of this one changes out of its all ones state. Then its TC output goes low, causing the latch to stop parallel loading and causing the communication wires leaving the processor to go low. One of those wires enters this processor on the PD input. At the next clock cycle, that low state is loaded into the D bit of the latch, turns on the right LED segment indicating a right pointing arrow, and prevents further parallel loading of the latch by forcing the TC output low. The result, then, is an indication on the LED for this processor that something happened to the right of it. Incidentally, when the TC output of this processor went low, so did the outgoing communication wires, so on the next clock cycle, the other neighbors of this processor will be activated just as described above. Now, how did all that get started? Well, the START input on one cell was momentarily grounded, causing the latch outputs to go to all zero, turning on all four LED segments indicating that propagation started there, and causing that cell's communication outputs to go low. It is actually a very simple process that each processor in the array must execute. There is no computation in the numerical sense

involved. Each cell simply passes on the propagating signal when it arrives, and records from which direction(s) it came.

When the propagation reaches the edges of the array, or can go no farther because of blocked cells, the action stops. What is recorded by the LEDs is actually the direction to go from each cell in the array to get back to the cell where it all started. The hardware has found the shortest path from the starting point to any other point in the array.

I designed this circuit purely for demonstration purposes. As such, tracing the path back is a visual process done by looking at the LED dispays. If automatic trace back were desired, the five bits in each processor would be accessed as five bit words by a general computer that would then consider the processor array as a block of smart memory. It is an easy task for an ordinary computer to decipher the bits from each processor to find the path desired.

Before proceeding, consider what has happened to the computational effort required to reach this result. Time complexity of the algorithm has been dramatically

improved. Now, rather than having a single processor advance the wavefront by stepping around the starting cell one cell at a time in an expanding spiral, the propagation takes place by activation of successive rings of processors surrounding the starting cell. At any given time, a number of processors directly proportional to the length of the path is actively working, rather than just one processor. The time required for the wavefront to expand out to the goal point is now directly proportional to the length of the path, not to the square of the length. Thus, the time complexity of the algorithm is now linear, not quadratic, with respect to the path length. This result is expected -- a linear number of active processors can do in linear time what one active processor can do in quadratic time.

The circuit described above is so simple that it seems natural to lay out several copies of it on a silicon chip. That is just what was done for the design of the MAZER chip.

6.2 The MAZER Chip

The first step in designing the MAZER chip was to develop a NMOS circuit that performed the function of the

demonstration circuit above. The stumbling block was the clocking scheme. Edge triggered latches are not as easy to come by in MOS as they are in TTL. Usually a set of multi-phase clocks is used to latch signals. This seemed to unnecessarily complicate the circuit, and a way around the problem was sought. The answer turned out to be easy. Just don't use any clocks!

On careful scrutiny of the operation of the demonstration circuit, one sees that clocking is really unnecessary. The only operation performed by each processor consists of waiting for the propagating wavefront to reach it, recording the direction(s) from which it came, and passing it along to its neighbors. One could imagine the array of processors as an array of mousetraps, each cocked and ready to fire. Each mousetrap is designed to fire as soon as any of its neighbors fire. Each mousetrap will store the direction from which its firing signal comes. At the end of the outward propagation process, which might always be allowed to propagate to the extremities of the array, the contents of each mousetrap cell's storage would then be the required arrows pointing in the direction of the shortest path from that cell to the start of the wave propagation process.

This is a good visualization of the way in which the MAZER works.

The clocking scheme that is so fundamental in synchronous digital design is thus seen in this case as merely an artificial limitation on the circuit imposed by restricted thinking. By allowing information to spread through the array at the full speed available in the hardware, unrestricted by clocking, the maximum power of the hardware is put to use in solving the path finding problem. The "path lenth" by which one path is judged shorter than another, is represented as the interval of time required for the wavefront to propagate from the initial cell to the goal cell, rather than as a precise number of discrete units of distance.

Figure 6-2 is a conceptual logic design of a simplified MAZER cell. Not shown are all the mechanisms for accessing the cell, blocking the cell so that it becomes part of a "wall" in the maze, causing that cell to be the starting point of wave propagation, etc., but the mousetrap characteristic is illustrated. After the reset line has gone high to make all the flip flop Q outputs low, all signals that cross the cell boundary are low, and the system is stable in this state. Now, if for some

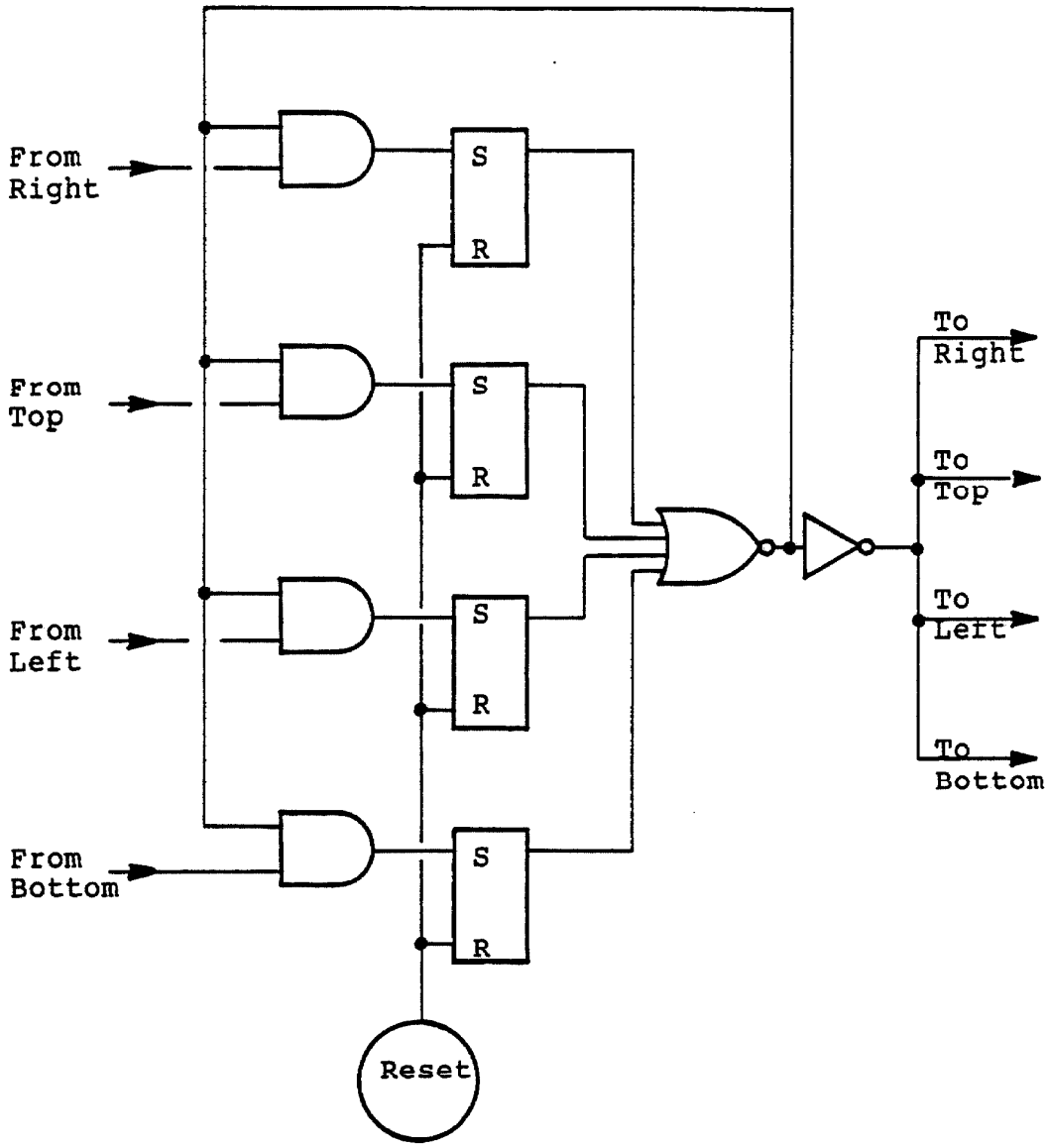


Figure 6-2. An electronic mousetrap.

reason one of the incoming signals goes high, the corresponding flip flop will be set. This causes the inputs to be disabled via the AND gates, and also causes the cell to generate a high going signal to each of its neighbors, triggering them in the same way. The flip flops remember from which direction the activation signal entered the cell, and reading them out by an accessing mechanism not shown gives the direction the maze solution takes as it passes through this cell.

Figure 6-3 is an actual schematic of a **MAZER** cell. Three of the AND gate/flip flop combinations of Figure 6-2 are seen here as transistor groups Q1-Q6, Q7-Q12, and Q13-Q18. The fourth direction is identified by the state where the cell has been triggered, and the other three flip flops are not set. The NOR gate and inverter are formed by Q19-Q25. Four bits of information are provided, as open drain outputs wire OR-ed with other cells on the chip. These bits are the three flip flop outputs plus a signal that indicates if the cell mousetrap has been "sprung". Transistors Q33-Q36 form a flip flop to store the blocked condition. ROW and COLUMN are addressing signals to select the cell for data readout, BLOCKing the cell to make it part of a maze wall, or STARTing the propagation process with this cell. RESET re-cocks the

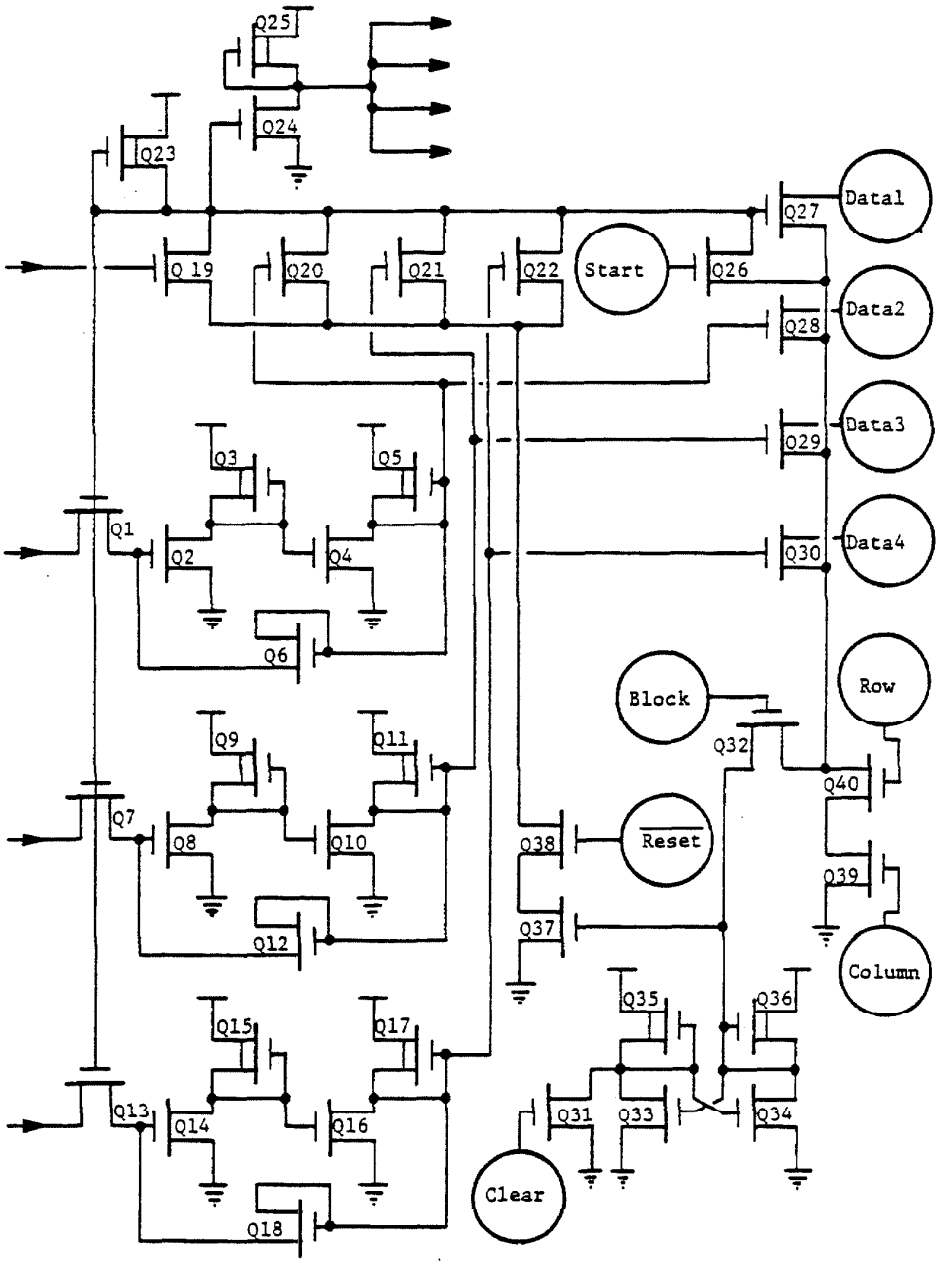


Figure 6-3. MAZER cell schematic.

mousetraps, but does not destroy the blocked condition in the maze wall cells. CLEAR unblocks all the cells in preparation for a new maze. The other signals are communication paths to adjacent cells.

The complete **MAZER** chip contains sixteen processors arranged in a four by four array. Larger arrays can be assembled by arranging **MAZER** chips themselves in an array. Four wires come off each edge of the chip for the purpose of communication to adjacent chips. There are 15 additional wires that come off chip for data and control. Four are for data outputs, four are for address inputs, two are for power, and five are for the control signals BLOCK, RESET, CLEAR, START, and CHIP-ENABLE. A plot of the **MAZER** chip is shown in Figure 6-4. The four by four array of processor cells can be seen, surrounded by the 31 pads. Designed with Caltech's conservative design rules, the chip measures 2241 microns square, or about 8100 square mils.

6.3 MAZER Chip Test and Characterization

Not until the **MAZER** was fabricated did I realize that there was an error in the design. When I tested the returned chips, they seemed to perform strangely. Finally

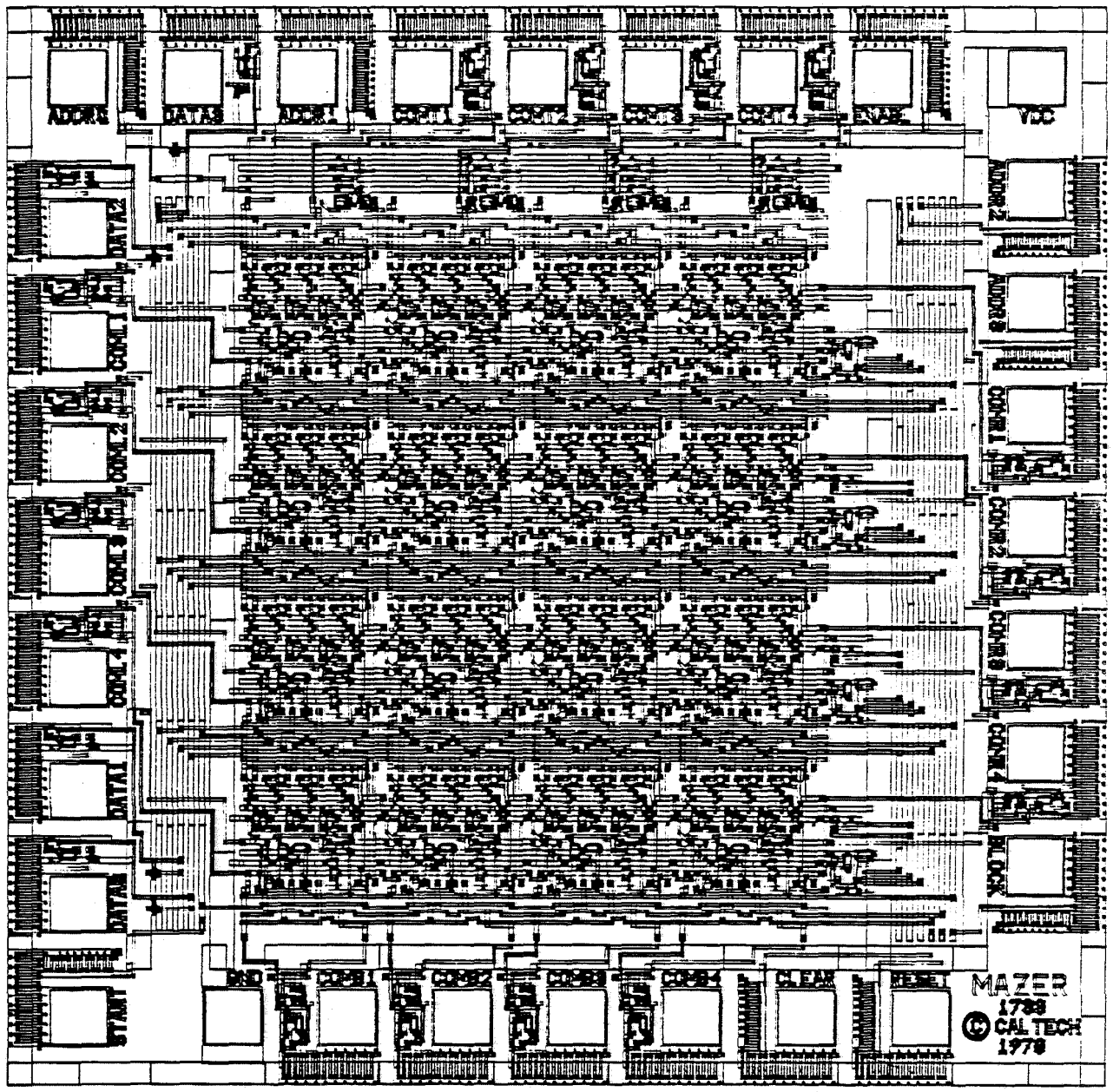


Figure 6-4. Plot of the MAZER Chip

I diagnosed the symptoms and found a basic bug in the MAZER processor circuitry. With the bug uncovered, I was able to circumvent the problem and continue to test the parts.

In order to understand the bug in the MAZER circuitry, re-examine Figure 6-3. The four data outputs of each of the sixteen processors, which come from the drains of transistors Q27 through Q30 in the figure, are bussed together onto four global data wires on the chip, DATA1 through DATA4. These wires run to the chip output pad circuitry, where there is a single pull up transistor on each of them. The important point to examine is the node in each processor that is common to the sources of the four transistors, Q27 through Q30. This node, which I will call the "enable node", is pulled low by the decoding transistors, Q39 and Q40, which are controlled by the addressing signals ROW and COLUMN. Since Q39 and Q40 are both turned on in only one of the sixteen processors, there should be a path from the enable node to ground only in that addressed processor. This prevents data in the non-addressed processors from pulling down on the data lines. However, things are not so simple. Suppose one of the data output transistors, say Q27, is on in the addressed processor, thus pulling down the DATA1 wire.

Now, suppose that in another processor, which is not being addressed, both Q27 and Q28 are on. Since DATA1 is being held low by data in the addressed processor, Q27 in the second processor provides a path to ground for the enable node in that second processor. As a result, Q28 in that second processor can erroneously pull down the DATA2 wire. Since the addressed processor should not pull down DATA2, the result is that incorrect data appear at the output of the chip.

Fortunately, it is possible to retrieve correct data despite that problem. In the above example, notice that DATA1 is pulled down first by the addressed processor in the normal way. The bad data do not start to pull down on DATA2 until DATA1 is down, and even then, the string of transistors doing the pulling on DATA2 is longer than normal. The result is that good data show up on the chip output pads about fifty nano-seconds before the bad data from the sneak paths ruin it. Latching the good data in an external latch at the right time retrieves the correct state of the internal bits.

The operations of STARTing propagation and BLOCKing the cell are also affected by the unwanted paths between enable nodes and ground. Luckily, because of the high

pull-up to pull-down ratio in the Q24/Q25 inverter, resulting in a very low switching threshold, STARTing can be performed normally. Apparently the sneak currents are low enough to prevent the inverter from switching in the non-addressed processors. However, I have been unable to individually BLOCK cells. The effect of blocked cells can nevertheless be tested by using the random distribution of blocked cells present after power-on. The CLEAR signal, which clears all blocked cells, operates normally.

In spite of the difficulties mentioned above, the test results are encouraging. All the arrows point correctly back to the cell where propagation starts. Some local asymmetries sometimes are present, indicating that propagation proceeds a little more quickly in some areas of the chip than in others, but this result is expected with the asynchronous scheme used, and is negligible anyway. Access time, from chip enable to data output, is around 100 nano-seconds, about normal for a chip of this small size. The **MAZER** chip is a successful solution to the problem of single layer path finding.

Chapter 7

TWO LAYER PATH FINDING

The fact that the **MAZER** is limited to single layer paths restricts its usefulness. The most immediate application for path finding hardware is in the area of printed circuit board design. However, single sided circuit boards are not very exciting. The step to two-sided boards dramatically improves wirability and board density. Adding even more layers to the board improves density still more, but the additional effort does not buy nearly as much as the move from one to two sides. Thus, there was great incentive to develop a two layer path finder, with the specific goal of producing a routing machine for two-sided printed circuit boards. This is the background for the design of the other integrated circuit to be discussed, the **PATHFINDER**.

At first glance, it would seem that one need only construct a circuit that forms the topology of two MAZER chips laid on top of one another, with an additional arrow bit in each cell to indicate travel from one layer to the other. This strategy would work, except that it lacks some properties that have been found very desirable in the two layer environment. In what follows, terminology of the printed circuit board world will be used, with the understanding that other applications, such as interconnect wiring on integrated circuits, would have analogous features and terminology.

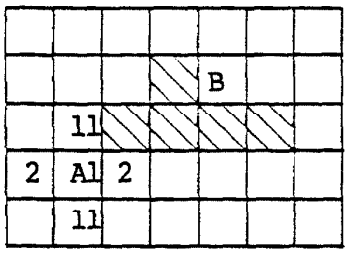
The first feature that would be missing in such a two-layer MAZER is the ability to block travel from one side of the board to the other independently from blocking travel through those cells without changing sides. Often it is desirable to prevent these holes in the board, or vias, from occurring in certain areas of the circuit board. Perhaps vias are to be allowed only on a tenth inch grid, for example. Furthermore, vias sometimes can affect more than just the cell in which they occur. A via in one cell may prohibit the placing of a via in an adjacent cell. For all these reasons, an additional bit is required for via blocking in any proposed two layer path finding system to make it useful.

The second missing feature is much more disturbing. Designers of two layer circuit boards have long realized that it is best for mostly vertical wire runs to end up on one side of the board, and mostly horizontal runs to end up on the other side. This helps to avoid unnecessarily blocking channels for future wires. The tendency of a wire to choose one side of the board or the other depending on its orientation would be completely lacking in a straightforward two-layer MAZER. Incorporating this preference into the basic path finding algorithm was an interesting problem, and the methods developed to solve it in both the traditional software implementations and the current hardware implementation will now be examined.

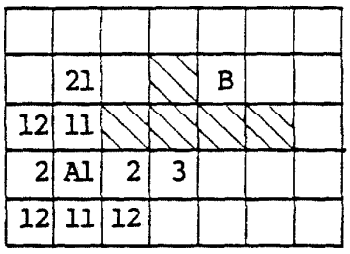
A way to achieve the wire location preference is to use a system of <u>costs</u> associated with travel from cell to cell through the array. Each cell still stores one integer, but rather than storing ascending ordinal numbers on successive wavefronts in the propagation phase, as in the original algorithm, each cell stores the accumulated cost for reaching that cell from the starting cell. Suppose C(a,b) is the cost for expanding the wavefront from cell a to its neighbor, cell b. Then as the wavefront passes from cell a to cell b, the number stored in cell b is the number stored in cell a plus C(a,b).

Since different costs might be encountered along different routes from the starting point to a given goal point, a number that has been previously stored in a cell might be overwritten if the wavefront reaches that cell from another direction with a lower cost than that achieved by the first contact with the cell. This is shown in Figure 7-1. Notice that the wavefront expands in exactly the same way as it did in the original algorithm, but now cells on the frontier are not necessarily all equally "distant" in terms of costs, as they were in the schemes described earlier. In the retrace phase of the algorithm, the numbers stored in the cells are used in a way similar to that described earlier. However, rather than searching neighbor cells for the next member in a reversed sequence, each step of the retrace involves searching for a neighbor of the current cell with a stored cost less than that of the current cell by the amount of the cost of propagating from that neighbor to this cell during the propagation phase. This does not necessarily result in the shortest path from point A to point B. What comes out instead is the least costly path between those two points, based on the cost function C(a,b).

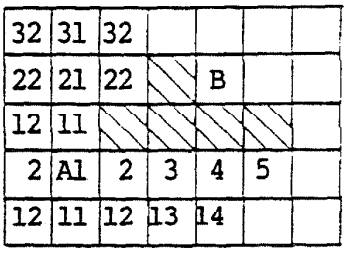
The simplest cost function normally used consists of only three distinct costs. One cost is used for



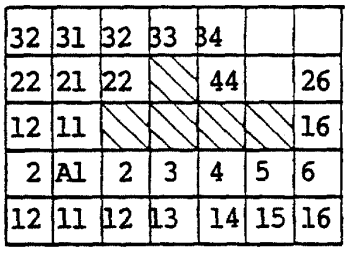
Just Starting



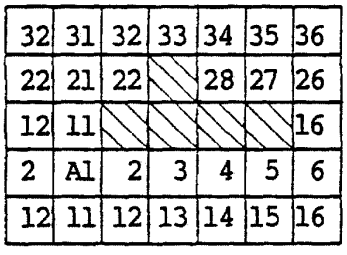
A little more



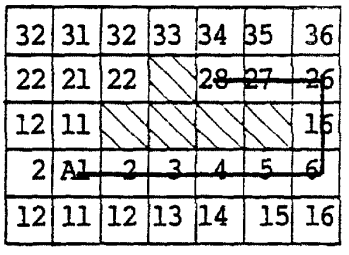
Still Farther



Not Done Yet!



Finished



Path Found

Figure 7-1. Path found by using a cost of 1 for travel in the horizontal direction and 10 for travel in the vertical direction.

travelling in the "easy" directions, north-south on one side of the board and east-west on the other side, a second, slightly higher, cost is used for travelling in the "hard" directions, east-west on the first side and north-south on the second side, and a third, even higher, cost is used for travel "through the board", from one side to the other. Fancier schemes are possible. These involve reduced costs for travel near to and parallel to the edges of the board to increase utilization of that area, increased costs near component pins to prevent blocking future access to those pins, etc. The task of developing a cost function tailor-made to a particular circuit board can become quite an art.

Note, however, that using this cost function as a solution to the two layer situation brings back the same problems the original algorithm had, namely an unbounded number of bits of storage per cell, and global distribution of a numerical cost function. If there was to be any hope of building a chip comparable to the **MAZER** for two layer circuit boards, these problems had to be eliminated. To accomplish this, the **MAZER** was re-examined.

The scheme of using arrows instead of numbers seemed to be the way to go to limit the number of bits per cell. Using this method, however, required that during the propagation phase the expanding frontier of the wavefront must include only cells that were equally distant in terms of cost from the starting point, unlike the above two layer approach. This seemed to be at odds with the uniform, diamond shaped wavefront propagation described above.

The solution to the costs problem was to control the speed of wavefront propagation from cell to cell, rather than let it go at gate delay speeds. Consider the simple three cost system described earlier. If propagation could be allowed to proceed quickly in the "easy" direction, more slowly in the "hard" direction, and even more slowly in the "through the board" direction, the wavefront would meet the requirement that all cells on the frontier would be at an equal "distance" in terms of cost from the starting cell. Imagine a system where north-south propagation is easy on the top of the board and hard on the bottom. On such a board, a wavefront propagating from point A to a point B directly north of point A will reach point B on top of the board first, and will thus store arrows indicating a path that travels on the top side of

the board back to point A. Similarly, if point B were to the east of point A, the wavefront, propagating more quickly in the east direction on the bottom of the board than on the top, would cause point B to store arrows indicating retrace along the bottom of the board back to point A. No matter where point B was, the arrival of the wavefront would store information describing the least costly path back to point A.

During the time this solution evolved, some redundancy in the storage used in the **MAZER** made itself known. If the propagation phase left an arrow in cell A pointing to a neighbor cell B, indicating that retrace should proceed in that direction, that implied that there would be no arrow in cell B that pointed to cell A. Since that particular combination, adjacent cells pointing at each other, would never occur, there must have been some redundant information stored there, implying that a reduction in storage was possible. This was accomplished by moving the location of the arrow bits from inside each cell to between cells. Since each arrow then served the two cells between which it lay, the total storage required for the arrows was halved.

With these new modifications to the basic Lee-Moore algorithm, it was time to start designing the two layer chip.

Chapter 8

TWO LAYER HARDWARE - - THE PATHFINDER CHIP

The obstacle in the design of the PATHFINDER chip was the method to use in controlling propagation speeds. What was required was a way to vary the speed over at least a ten to one range in each of three directions, the "easy" direction, the "hard" direction, and the "through the board" direction. Also, the circuitry could not be overly complex, nor could it involve many wires to the global environment. However, the required speed settings were related to the cost function described earlier. The cost function was something that was set by little more than educated guessing and experimentation. There was nothing very critical about the exact values of the costs. Only approximate settings were required. All of these considerations led the design away from a digitally controlled speed system, and toward a hybrid system.

The method employed relies heavily on the dynamic charge storage abilities of MOS circuitry. Figure 8-1 shows the set up for a simplified, one layer cell with its surrounding arrows, not showing the blocking or accessing circuitry. Each cell contains a capacitor of about 5 pf. Before the start of the propagation phase, the capacitors are all precharged by means of the precharge transistor. With all the capacitors charged, all the arrow flip flops have both outputs held low. To start propagation at a cell, that cell's capacitor is discharged. That action releases one side of the arrow flip flops surrounding that cell, causing those arrows to "point" to that cell with the discharged capacitor. The high outputs of the arrow flip-flops then enter the neighbor cells, and begin discharging the capacitors there at rates determined by the voltages on the gates of Qa and Qb. When those capacitors are completely drained, the arrows surrounding those cells flip to point to the newly discharged capacitors, and the arrow outputs begin discharging capacitors in <u>their</u> neighbors. As the wavefront of activity propagates out, cells behind the frontier have completely discharged capacitors, cells ahead of the frontier have fully charged capacitors, and cells on the frontier have capacitors that are in the process of being discharged.

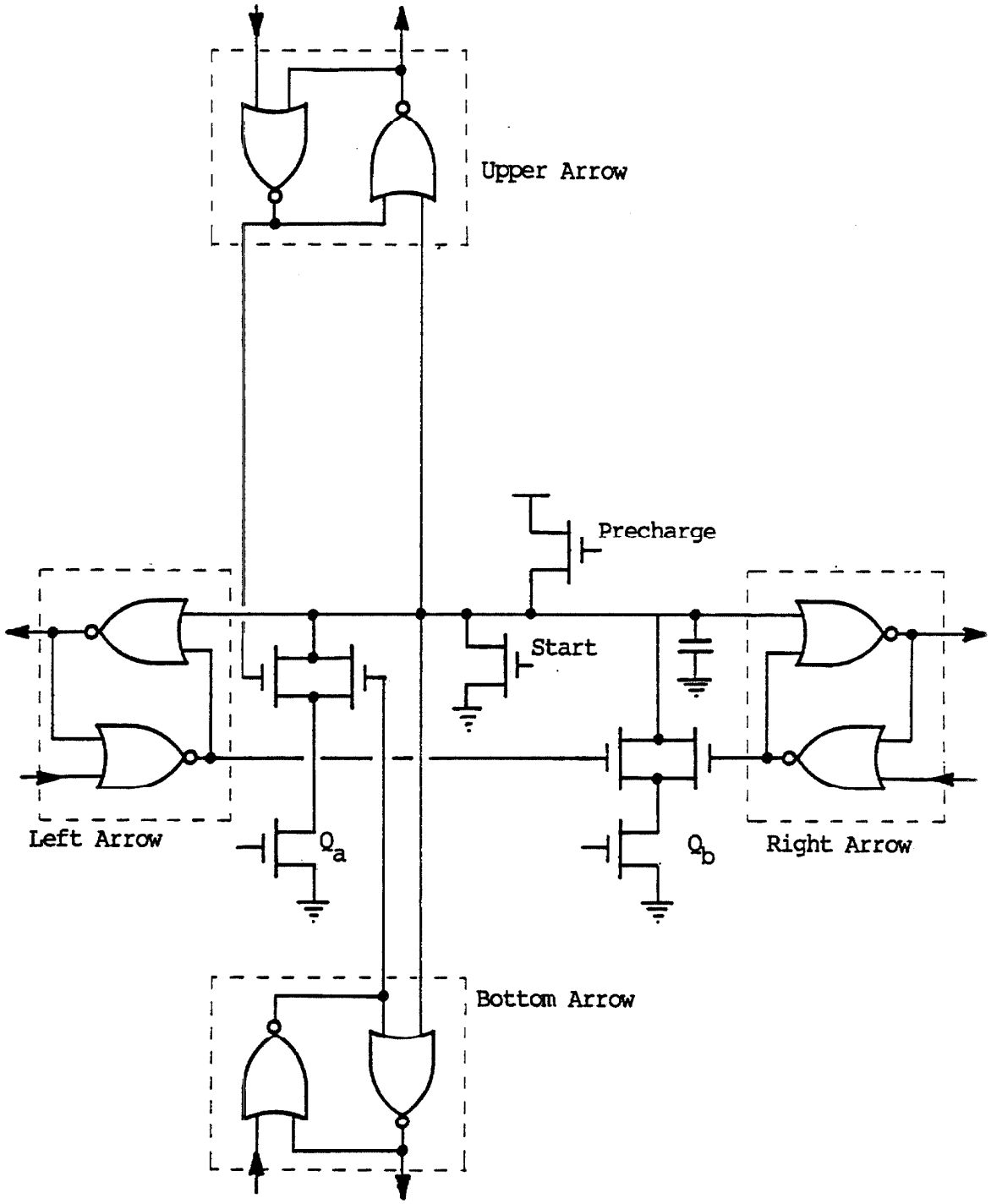


Figure 8-1. Simplified PATHFINDER Cell

The time required, and thus the cost, for propagating through a cell depends on the rate at which the capacitor is discharged, which, in turn, depends on the voltages on the gates of Qa and Qb. The direction from which the wavefront approaches the cell determines which path to use in discharging the capacitor.

A feature included on the chip allows a small amount of local control over the cost function to modulate the overall three costs described above. This consists of an additional pF or so of capacitance that can be switched on in parallel with the main capacitor in each cell. The time for propagating through a cell, and hence its propagation "costs", can be increased by connecting its extra capacitor before precharge and leaving it connected through propagation. The cost can be decreased by connecting the extra capacitor after precharge is over and disconnecting it again before propagation starts. These capacitor connections are switched on a cell by cell basis, controlled by a single bit in each cell. This makes it possible to increase costs near component pins, or to decrease costs near the board edges, etc., to reduce or increase the tendency for wires to end up in those areas. If circuitry had been included to discharge the extra capacitor when it was disconnected from the main

one, additional levels of cost could be obtained by repeatedly connecting and disconnecting the extra capacitor between precharge and the start of propagation to remove more and more charge from the main capacitor, and thus reduce its discharge time. However, that extra feature was not included.

Figure 8-2 is a schematic of a two layer PATHFINDER processor, containing circuitry for the cells on both sides of the board as well as the arrows between them. The upper arrow and right hand arrow for each cell are arbitrarily assigned as belonging to that cell, while the lower and left hand arrows are considered to belong to the neighbor cells in those directions. The control storage bits are shown as boxes for simplicity. Actually the five arrow bits and the four control bits make up a nine bit word of what amounts to a standard static memory system, using the usual six transistor cell. Not shown are the two transistors that selectively link the flip flops to the word lines that run through all the bits, nor the select lines that control the gates of those transistors to do the addressing. Instead, the storage bits are shown located in reasonable places on the schematic to suggest their function in the circuit.

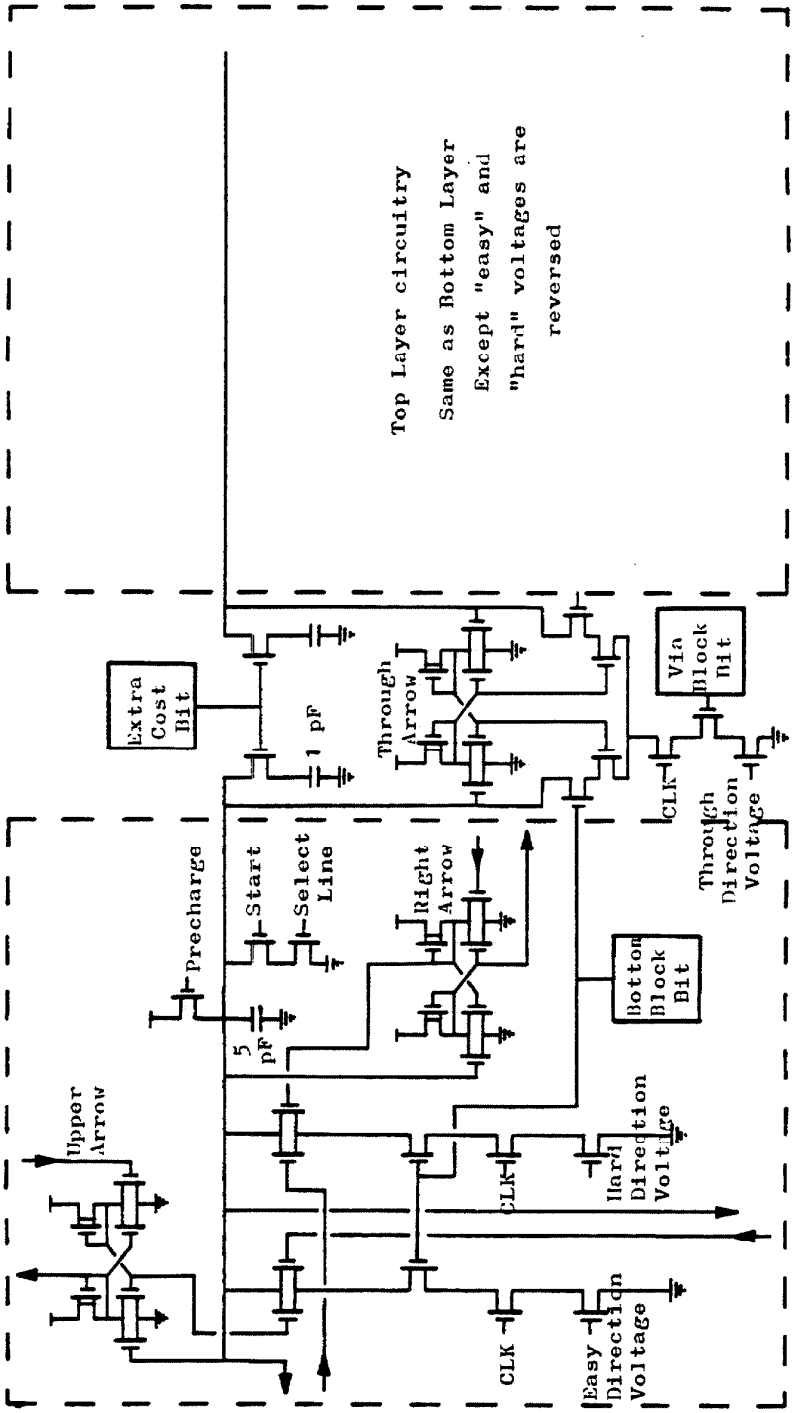


Figure 8-2. Schematic of PATHFINDER processor. Bit reading and writing mechanisms are not shown.

The circuit works just as described above for Figure 8-1, with the addition of the blocking controls and the switch to two layer operation. Having two layers merely means that three paths are present for discharging the capacitor, each controlled by a transistor whose gate voltage determines how quickly the capacitor is discharged through that path. The blocking control flip flops merely inhibit the appropriate discharge paths to prevent the discharge of the capacitor under the conditions that are to be blocked.

Figure 8-3 shows a plot of the metal layer of the **PATHFINDER** chip. The chip contains a four by eight array of two layer processors. As with the **MAZER**, the large processor arrays of several hundred processors on a side that are needed for useful printed circuit board work are built up by assembling **PATHFINDER** chips themselves in an array. Forty-eight of the seventy pads are devoted to chip to chip communication within the large array. The remaining pads consist of nine address pads, two power pads, two data I/O pads, and nine control pads. Chip size is 3750 by 4875 microns.

One signal in the **PATHFINDER** cell's schematic remains to be defined. This signal, labelled CLK in Figure 8-2,

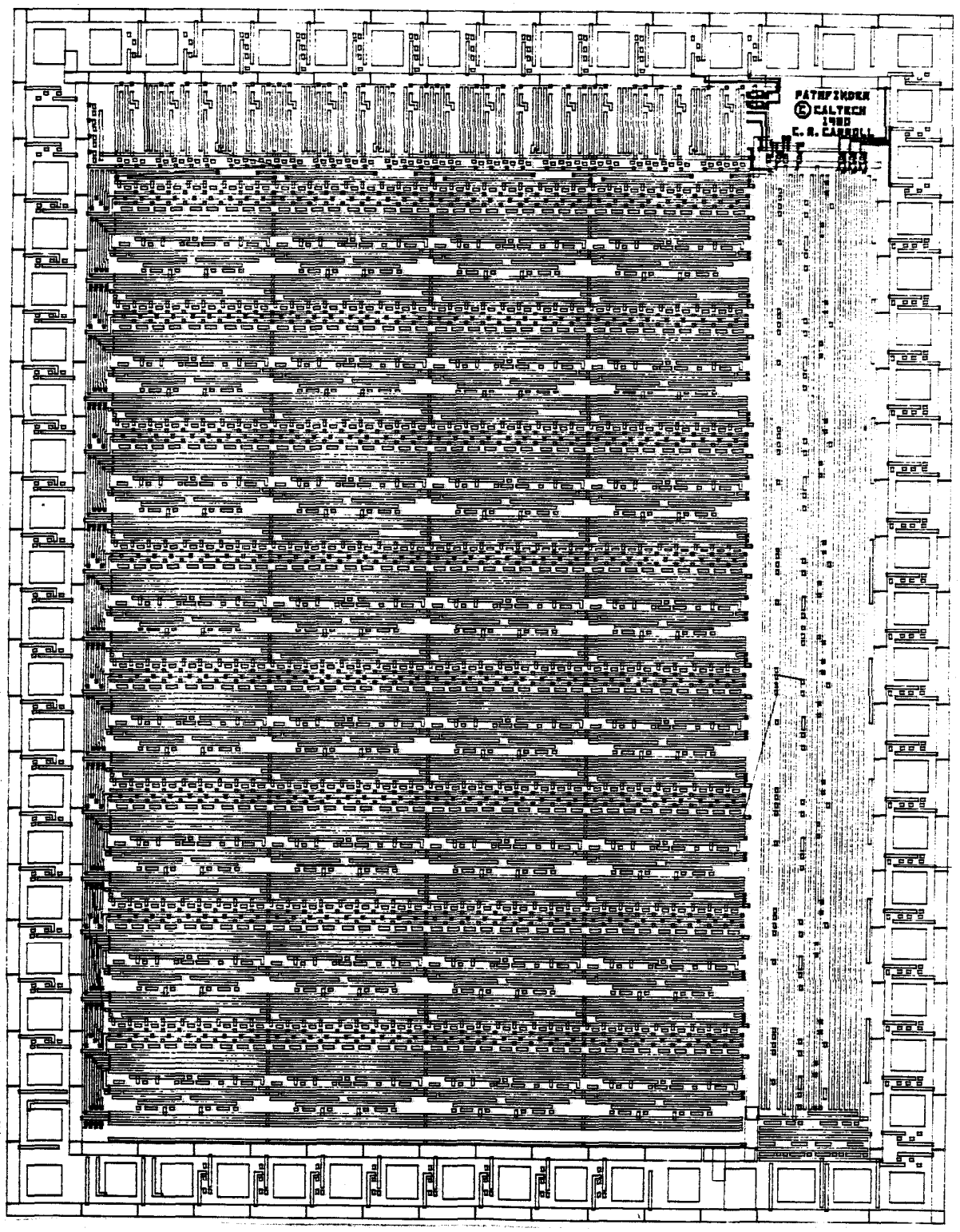


Figure 8-3 . The PATHFINDER's Metal Layer.

appears on the gate of a transistor in series with each of the discharge paths in every cell. Only when CLK is high can the discharge action occur. When CLK is low, all activity is suspended. The need for this signal stems from a fundamental difficulty that hybrid system designers must face. The next chapter defines that difficulty and shows how the CLK signal and other facets of the PATHFINDER's design evolved to avoid it.

Chapter 9

DEALING WITH NON-UNIFORMITY

Any processing system must eventually be implemented with real hardware, and in any real implementation, there will be parameters that will vary unintentionally. One can never count on a perfectly uniform environment in which to build and operate his system. Part of any processor design thus must include concern over the sensitivity of the design to variations in various parameters in the fabricating and operating environment.

Digital elements are very insensitive to non-uniform conditions as a rule. Digital representations of data provide a range over which the physical quantities may vary without changing the value of the data represented, just as the abacus could tolerate slight displacements of its beads without losing its data.

Analog elements, however, generally require tight restrictions on variations in their environment. Any change, no matter how slight, which affects a data-representing quantity will introduce an error in the data that will be carried along into succeeding calculations. Proper operation of an analog system generally requires careful control over the fabrication parameters and operating conditions.

A hybrid system, such as the **PATHFINDER**, by definition includes some analog processing, and thus demands that one consider its sensitivity to environmental variations. In the **PATHFINDER**, there are two areas that require careful control. These areas are the generation of the cost-setting voltages that are applied to the gates of the transistors at the bottom of the discharge paths, and the uniformity of the delays in cell-to-cell communication throughout the array of cells. The **PATHFINDER's** design includes special features to handle both of these issues.

Controlling the cost-setting voltages is the first sensitive area of the design. We need to generate three voltages to be applied to the gates of the transistors at the bottom of the three discharge paths in each cell. The

object is to control the current in the discharge paths, and thus to control the time required, or cost, for propagating the wavefront through the cell. The cost system demands that the three costs be uniform from cell to cell throughout the array. Thus, one's first reaction is to distribute the same three voltages to each cell, thereby setting the gates of the three controlling transistors at the same voltage in each cell, and thus allowing the same currents in the three discharge paths in each cell.

Unfortunatly, the discharge currents, and thus the costs, depend not only on the gate-to-source voltage of the controlling transistors, but also on the threshold voltage of these transistors, which is set during fabrication. Variations in the transistor's threshold voltage would result in different discharge currents, even with the same voltage applied to the gates. Variations in threshold from one cell to the next on the same **PATHFINDER** chip are small enough to be ignored. However, the **PATHFINDER** system must be assembled from many individual chips, which may have been fabricated at different times or on different lines, and will very likely have different threshold voltages. Thus, although the same three controlling voltages may safely be distributed to all

cells on one chip, the voltages that result in equal costs will not be quite the same from one chip to the next. It would be very awkward to set three separate voltages independently for each of the many PATHFINDER chips in a system, so some scheme is needed to reduce the sensitivity of the cost control system to transistor threshold variations.

The PATHFINDER's solution to this problem is shown in Figure 9-1. That figure diagrams two PATHFINDER cells, and shows the cost-setting interconnections. All of the resistors that have the same label in the figure have the same value. The scheme used for setting the costs is to set the discharge currents directly and not worry about generating the control voltages themselves, using a circuit known as a current mirror. The resistors in the figure bias their corresponding transistors on the chips to some quiescent operating point, where the currents that pass through the transistors are determined by the values of the resistors and the voltages across those resistors. The voltages at the high end of all the Rl's in the system are the same. The voltages at the low end of the Rl's would all be the same if all the transistors had identical characteristics. In fact, the transistors differ slightly, so the voltages there will differ slightly. It

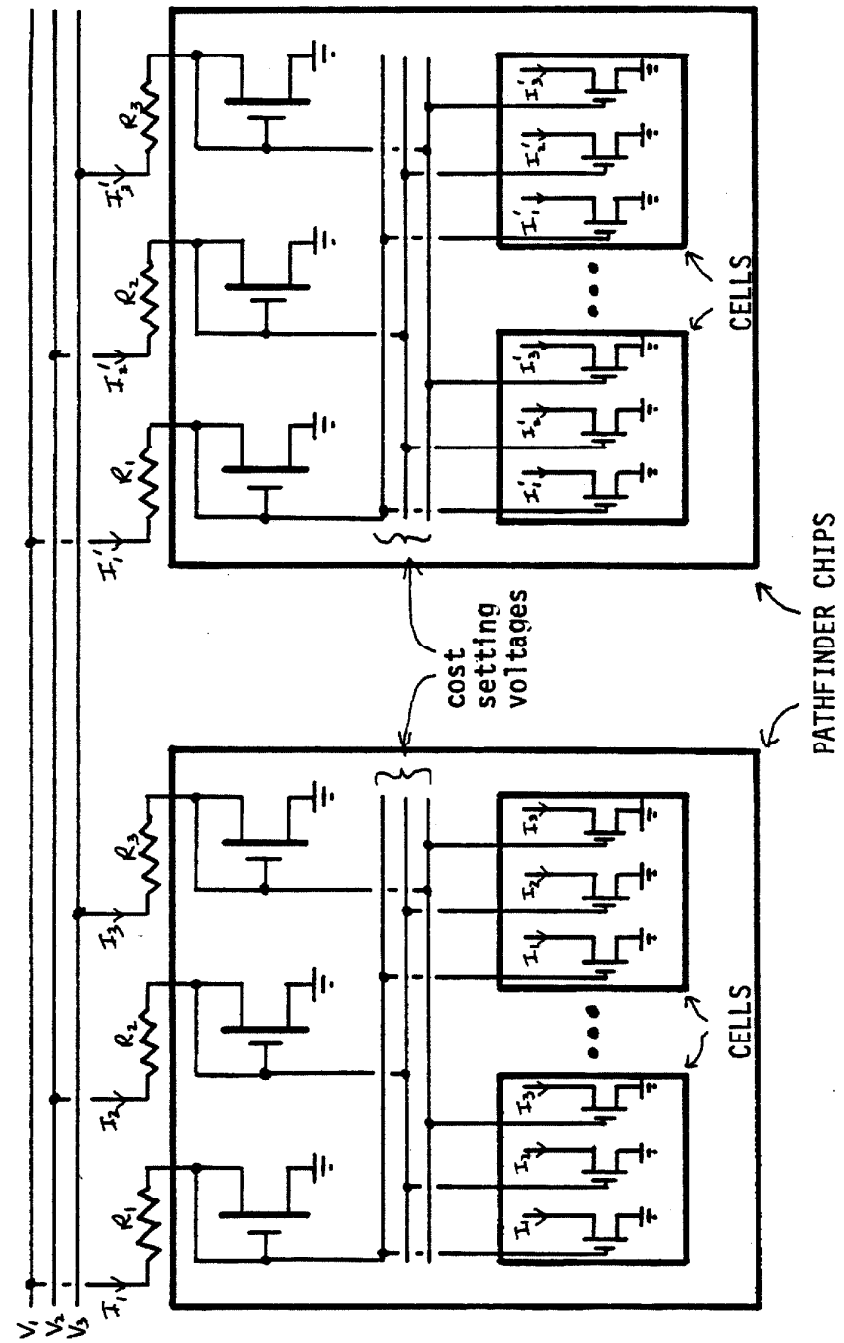


Figure 9-1. The current mirror cost-setting arrangement. Assuming that all transistors on the same chip are identical, then identically labelled values above will be equal, and values whose labels differ only in a prime will be nearly equal, despite chip-to-chip variations in transistor characteristics.

is to that difference that we must reduce the circuit's sensitivity. Note that as long as V1 is well above the threshold voltage of the transistors, then the voltages across the R1's will be dominated by V1 and only slightly modulated by the varying voltages at the low ends of the resistors. Thus, the currents through the R1's will all be nearly equal, varying only slightly with the variations in transistor threshold. The voltage at the bottom of the R1 connected to each chip becomes the control voltage for one of the costs on that chip. Any transistor on that chip whose gate is connected to that voltage will be biased to the same point as the current mirror transistor, and will thus carry the same current as that flowing in the corresponding R1 as long as the transistor operates in its saturation region. Thus, the discharge current for the path controlled by that voltage will be equal to the current flowing in R1 for that chip. Since all the currents in the R1's are nearly identical, the discharge currents for that path in all the chips will be nearly identical, practically independent of variations in the transistor characteristics from chip to chip. This is the desired result.

In practice, V1, V2, and V3 may all be the same voltage, say five volts, and the three costs would be set

by choosing different values for the R1's, R2's, and R3's to set the different current levels. This scheme would work well for a system in which the costs were fixed. To vary the cost structure, however, would require changing the resistors connected to every chip in the system. To avoid such an awkward situation, one could keep V1, V2, and V3 separate and vary those voltages to modulate the global costs. As long as those voltages are well above the threshold of the transistors on the chips, so that the voltages across the resistors, and thus the currents through them, are dominated by those voltages V1, V2, and V3, the discharge currents, and thus the costs, will be quite insensitive to chip-to-chip varations in transistor threshold.

The other critical aspect of the **PATHFINDER's** design has to do with the uniformity of the interconnection of the cells in the array. The analog cost scheme depends on a uniform delay in propagating the expanding wavefront from cell to cell in the array. The current mirror solution above ensures that the discharge of the capacitors occurs uniformly. However, there is an additional delay experienced in propagating the communication signals from one cell to the next. This extra delay contributes to the cost function and if it is

not uniform throughout the array of cells, it introduces a non-uniformity in the costs.

The regularity of the cells on the **PATHFINDER** chip itself imposes a natural uniformity on the communication structure. If all the cells in the system could be fabricated on a single chip of silicon, there would be little problem. However, the **PATHFINDER** system builds large arrays of cells by interconnecting many **PATHFINDER** chips, and the boundaries between chips lead to a distinct non-uniformity in the communication time. Signals will propagate much more quickly between adjacent cells on the same chip than they will between adjacent cells that happen to exist on different chips. This increases the cost for the communication paths that cross chip boundaries. The **MAZER** chip is plagued by this problem as well, since it also finds paths based on the time needed to propagate a signal from one point to another. The effect of this problem is shown in Figure 9-2. The minimal cost path between A and B in the Figure may well be the solid line rather than the dotted line, because the solid line crosses fewer chip boundaries.

One way to minimize this problem would be to reduce the discharge currents in the cells so that the capacitor

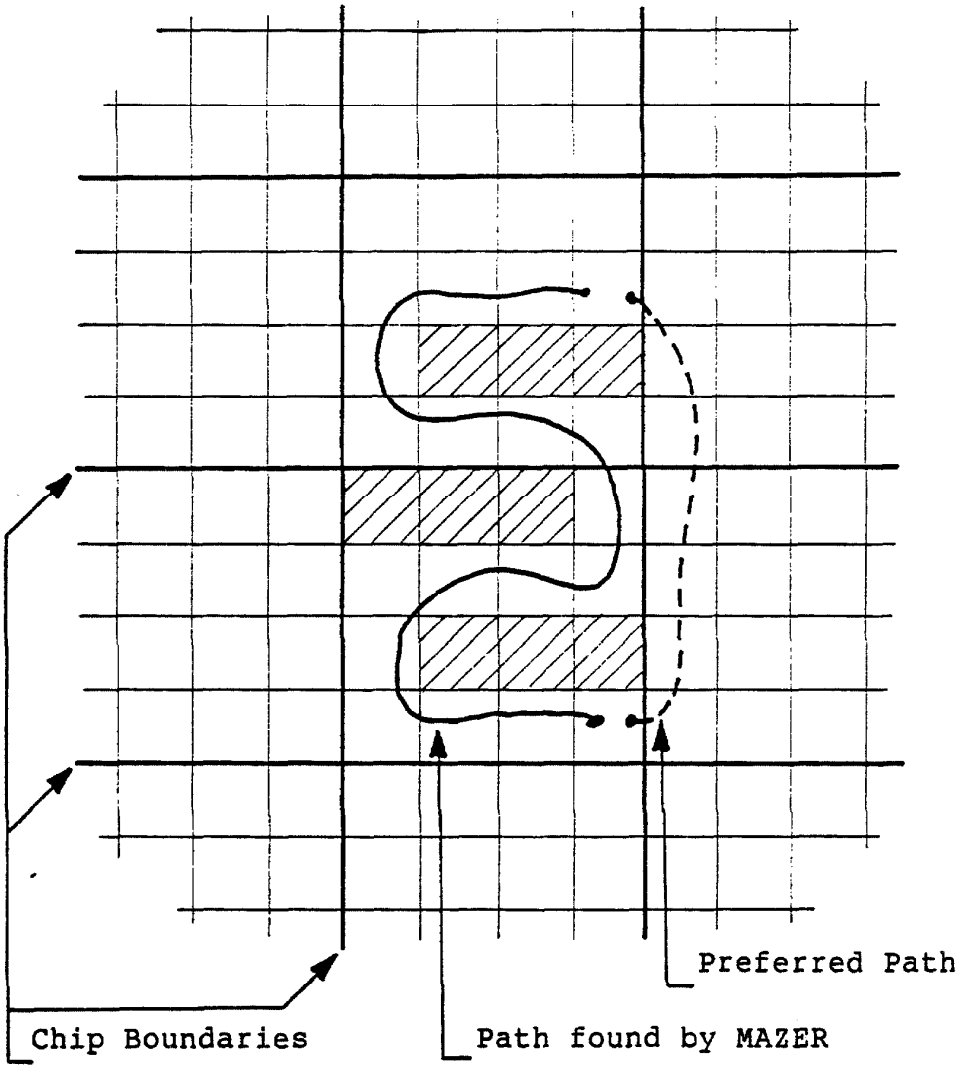


Figure 9-2 . The problem posed by chip boundaries in large arrays.

discharge time is much longer than the chip-to-chip delay time. With that approach, the percentage error in the propagation delay from chip to chip would be 100*D/T %, where D is the additional delay encountered in crossing a chip boundary, and T is the discharge time of a capacitor. In the **PATHFINDER** chip, the shortest possible capacitor discharge time, corresponding to the lowest cost, is about 200 nano-seconds. If the delay in changing chips is 100 nano-seconds, a 50% error in cost results, which is unacceptable. By using lower currents, the error can be reduced, however. Only a 10% error would result if the shortest discharge time were limited to 1000 nano-seconds.

The method used in the **PATHFINDER** for controlling this problem involves the CLK signal mentioned at the end of the preceding chapter. When CLK is high, discharge occurs normally. When CLK is low, all discharge paths are interrupted and the action stops. The technique here is to allow the capacitors to discharge only a little at a time by letting CLK be high for only short periods, followed by a "settling time" equal to D above, when CLK is off and any signals crossing between chips have time to get there. Suppose that each discharge period is divided into N active segments separated by this settling time.

If those segments of the discharge period are shorter than the settling time, which would generally be the case, then a cell on one chip watching for a capacitor in an adjacent chip to discharge would observe that event during the settling time following the active period in which the event actually happened. That observing cell then would react to the event as if it happened at the end of the active segment of time even though it actually may have happened at any time during the active segment of discharge. The average error, then, in the time to propagate the wavefront between these two cells on different chips would be one half of one segment of the discharge time, or T/2N, giving an average percentage error of 50/N %. For a 10% error, we would need to break up the discharge period into five segments, each followed by a settling time period. The total time, then, for propagating a lowest-cost signal from cell to cell in the **PATHFINDER** would be the shortest capacitor discharge time, 200 nano-seconds, plus five settling times of 100 nano-seconds each, or 700 nano-seconds total. Comparing this with the 1000 nano-seconds required for the simpler solution mentioned above, one sees that this second solution allows faster operation in this case. For a system in which the delay from chip to chip was a smaller fraction of the capacitor discharge time, the situation

may reverse. Note that the **PATHFINDER** can operate in either mode by either pulsing the CLK signal as described above, or merely keeping CLK high always.

The **PATHFINDER** uses the solutions described above to attack the problem of non-uniformities in the system. They are only approximate solutions. One can never reduce to zero the effects of the chip boundaries, for example. Furthermore, other effects such as variations in capacitance and sheet resistance of the chip materials introduce small variations of their own. Another item neglected is the small variation in transistor threshold voltage between the cost-setting transistors on the same chip. No matter how hard one works, there will always be some non-uniformity remaining in any real implementation to upset the operation of any analog processing system, including the analog portions of hybrid systems. Taking steps like those described in this chapter, however, can reduce the effects of those variations to acceptable levels.

Chapter 10

FLAWS IN THE PATHFINDER

The PATHFINDER chip was included on the MPC380 run managed by Xerox PARC in the spring of 1980, and was also included on the M08B Mosis run managed by the Information Sciences Institute. From the two runs, I have received approximately twenty five copies of the chip. Four of these chips apparently contain no processing errors and perform as expected. Faults in the bad chips range from stuck bits to malfunctioning address decoders to complete failure, perhaps in the output buffers. Some of the chips have capacitors that fail to hold charge long enough to be useful. The capacitors in the good chips, though, hold their charge long enough to keep the arrows "balanced" for about twenty seconds when carefully shielded from light. This is several orders of magnitude longer than required for successful path finding.

The chips have passed through several quantitative tests. Access time from chip enable to data out is around a micro-second, which is acceptable, though not noteworthy. An important item of interest is the operation of the cost-setting current mirrors. These perform very well. The range of control is excellent, with the normal external operating current between 100 and 1000 microamps.

I have assembled a small microcomputer system to test the **PATHFINDER** chips and to operate them as a pathfinding system. I have written a true path finding program for the microprocessor that uses one **PATHFINDER** chip to find paths through a four by eight grid. The program allows the user to set up any initial combination of blocked cells and blocked vias, start propagation from any cell in the array, and trace a path back to that starting cell from any of the other cells. The program displays the resulting path as well as the status of the control bits in the **PATHFINDER** chip on a terminal screen. In this implementation, three potentiometers, connected to the three current mirror pads on the chip, are used to set the three global costs. The program can demonstrate the effect of changing costs on the path found between two points in the grid. For example, the user can cause the

path to either skirt around barriers between the two endpoints, or to form vias and go over or under the barriers, depending on the settings of the three potentiometers.

Experimenting with a system even as small as this one can provide some insight into the operating characteristics of a hybrid system such as the **PATHFINDER.** A digital-looking processor with three potentiometers on the front is a strange sight. The system does a good job of demonstrating the path-finding abilities of the **PATHFINDER.** After experimenting with this system for some time, though, a few anomalies emerged and pointed to some deficiencies in the **PATHFINDER's** design that should be changed in a second generation version of the chip. As it stands, the chip does not always find the least costly path, but rather will sometimes choose a path with a cost slightly higher than minimal. There are three faults, each of which independently causes some of these incorrect results.

The first design flaw has to do with how the capacitors are discharged. If the wavefront reaches a cell from, say, both the north and east directions simultaneously, the capacitor will be discharged at a rate

equal to the sum of the easy and hard direction rates. This is because more than one of the three discharge paths is allowed to drain charge from the capacitor. The result of this flaw is shown in Figure 10-1. Instead of assuming the shape of an expanding diamond as it propagates out from the initial cell, the wavefront expands in a nearly rectangular shape. One can understand this by recognizing that the frontier cells on a part of the wavefront propagating diagonally through the array will fire more quickly than normal, resulting in the frontier at that point bulging out slightly from where it should be. If one started with a diamond shaped frontier, the sides of the diamond would expand more quickly than the points of the diamond, and the frontier would gradually evolve into the nearly rectangular form shown in the figure. Since the algorithm says that all cells on the frontier at any given time have the same accumulated cost back to the original cell, this flaw reduces the effective cost of paths to points near the corners of the rectangular propagation pattern. In the printed circuit world, this would have the effect shown in Figure 10-2. Although Path A and Path B in the figure should be equally costly, the **PATHFINDER** may claim that A is less costly, and thus cause a preference for Path A over Path B that should not exist. The effect is minimal, but annoying. As more

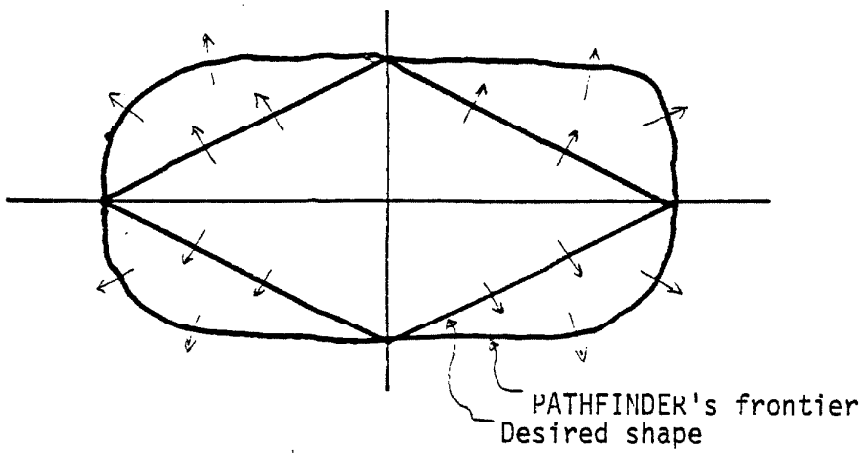


Figure 10-1. The shape of the expanding wavefront frontier in the PATHFINDER compared with the desired diamond shape.

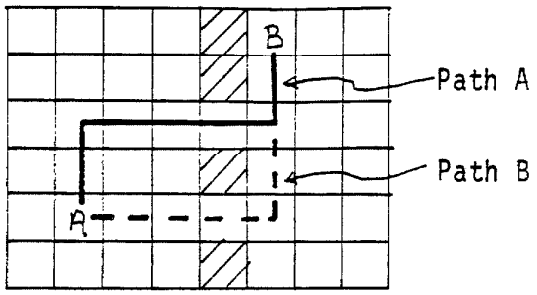


Figure 10-2. Due to the frontier shape displayed above, the PATHFINDER would prefer path A over path B.

wires are placed, the effect diminishes, since the wavefront propagates down skinny alleys where there is no chance of forming any two-dimensional shape, either diamond or rectangle.

A fix to this flaw is easy. All that is required is a protection on the discharge paths, so that when one is turned on, the others are blocked. The discharge path carrying the greatest current, representing the lowest cost, has the highest priority. Using such a scheme and fixing the other flaws to be discussed will achieve the normal, diamond shaped wavefront frontier, as desired.

The second flaw in the **PATHFINDER** design has to do with timing and the incorporation of the analog costs, and provides an important lesson in the study of hybrid systems in general. Figure 10-3 shows the conceptual cell logic diagram of Figure 6-2, with the addition of the delay controlled by the analog cost variables. The design essentially consists of a five input one output asynchronous state machine with five state variables, the flip flop outputs. The **PATHFINDER** positions the delay inside the feedback path, as shown in part a of Figure 10-3. When one input goes high, the corresponding arrow flip flop is set and the delay starts. If, before the

delay times out, another input goes high, that arrow flip flop is also set. Other inputs to the cell are blocked only after the delay is over. There is nothing to distinguish which of several set arrow flip flops was set first and which were set by inputs arriving later but before the delay timed out. This effectively reduces the cost of the paths generating the later inputs to equal the cost of the path that generated the initial input to the cell. The machine is tricked into thinking that all the set arrow flip flops resulted from inputs that arrived at the same time, when really they did not. The only way to avoid confusing the machine in this way is to keep the delay induced by the analog inputs out of the feedback path in the state machine.

Given that we have to move the delay in order to fix this flaw, the question becomes where do we move it? The obvious choice is shown in part b of Figure 10-3, at the output of the cell. This position, however, also is not ideal. The amount of delay here is determined by which arrow flip flops are set and the corresponding costs associated with them. The delay in passing an advancing wavefront through the cell from input to output is correct and only the first input to arrive is able to set its arrow flip flop so no confusion results. However, the

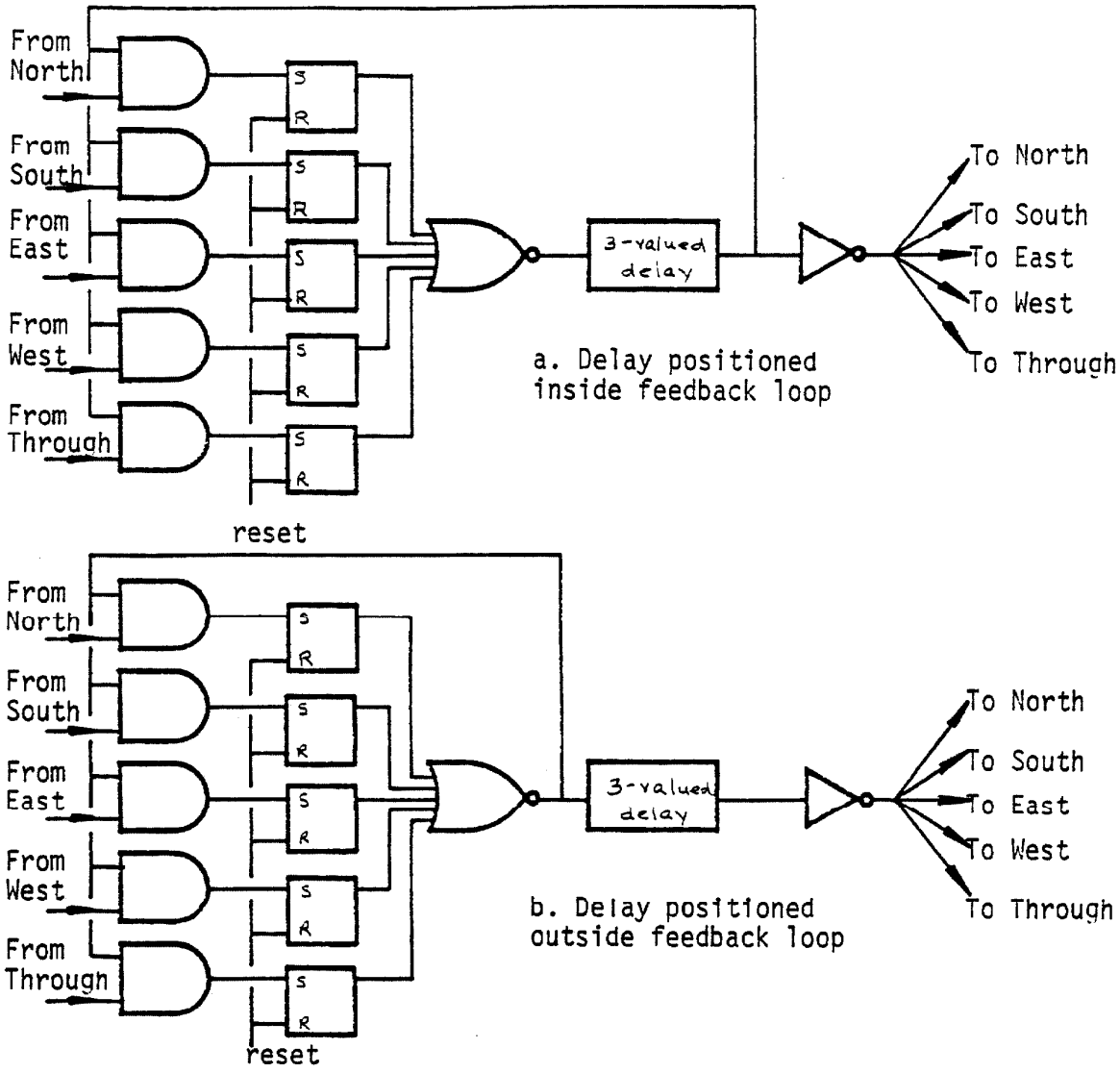


Figure 10-3. Conceptual PATHFINDER cell logic with the analog-controlled delay positioned differently.

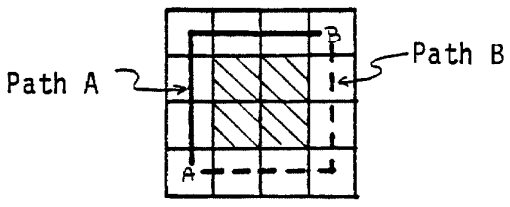


Figure 10-4. If propagation starts at cell A and horizontal travel is cheaper than vertical travel, the circuit of Figure 9-3b will prefer path A over path B.

arrows are set on the basis of the inputs without taking the delay in this cell into account. The effect of this is to make the cost of travelling in the cell where the path ends equal to zero. This results in the situation shown in Figure 10-4, where path A is preferred over Path B, even though both should be equally costly.

The effect of this flaw in wire routing is noticeable only on very short paths, and thus is not especially important in this application, but the theoretical implications of this problem are more extensive. Careful study of the path finding algorithm reveals that a path accumulates cost not by travelling <u>through</u> a cell, but rather by travelling <u>between</u> cells, from one cell to another. Thus, one cannot implement the algorithm perfectly by positioning the costs, or delays, within the cells at all. The only way to correctly match the hardware to the problem is to place the delays <u>between</u> the cells, on the communication paths, as shown in Figure 10-5. The delays pictured there are bi-directional elements that delay signals passing through them in either direction by the same amount of time. Only by associating the costs with the communication in the processing can this second flaw in the PATHFINDER be corrected.

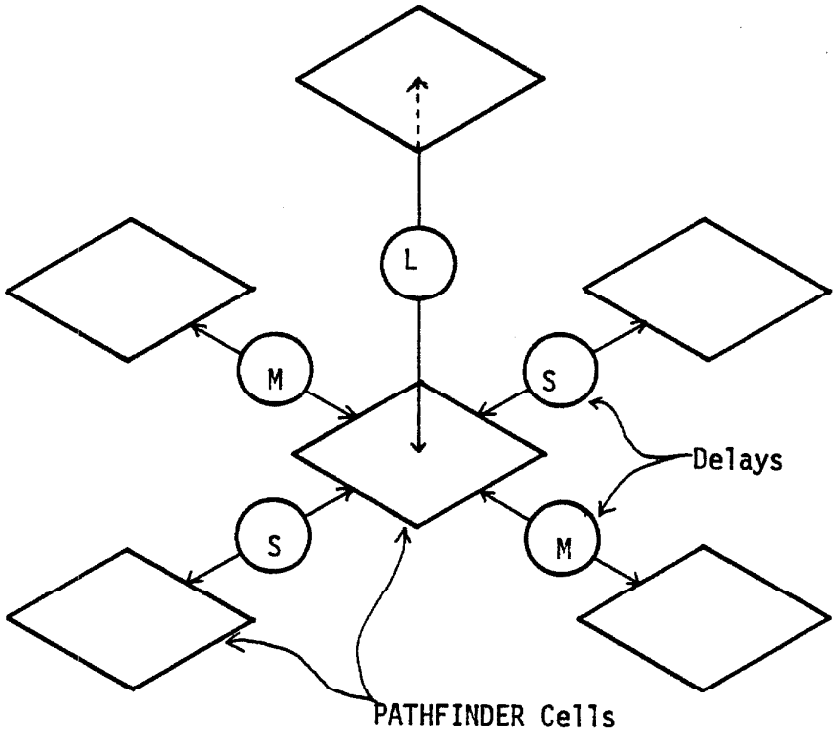


Figure 10-5. Implementing costs with delays on the communication paths linking each cell with its neighbors. The circles indicate long (L), medium (M), and short (S) delays.

The third flaw in the **PATHFINDER** design stems from the attempt to reduce the amount of storage based on the assumption that arrow bits within the cells are redundant, as mentioned in Chapter 7. Given two adjacent cells A and B, it is true that the wavefront will never propagate both from A to B and from B to A, with arrows pointing at each other. However, there are <u>three</u> situations that are possible. Obviously, the wavefront can propagate from cell A to cell B or from cell B to cell A. It is also possible that the wavefront does not pass between the cells in either direction, and it is important to note that. Consider the case shown in Figure 10-6 with the arrows implemented as in the **PATHFINDER** chip. Here a plane wave is propagating along the x-axis of the array towards our cell A. The horizontal arrows all will point to the west, as required, but the vertical arrows will end up pointing randomly either north or south. If the vertical arrows end up all pointing south, say and the retrace from cell A happens to start by checking the south-pointing arrow, the resulting path will head south, and continue south, instead of heading west, in the direction from which the wavefront actually came. A third arrow state is required to indicate this case when the wavefront does not pass between two adjacent cells.

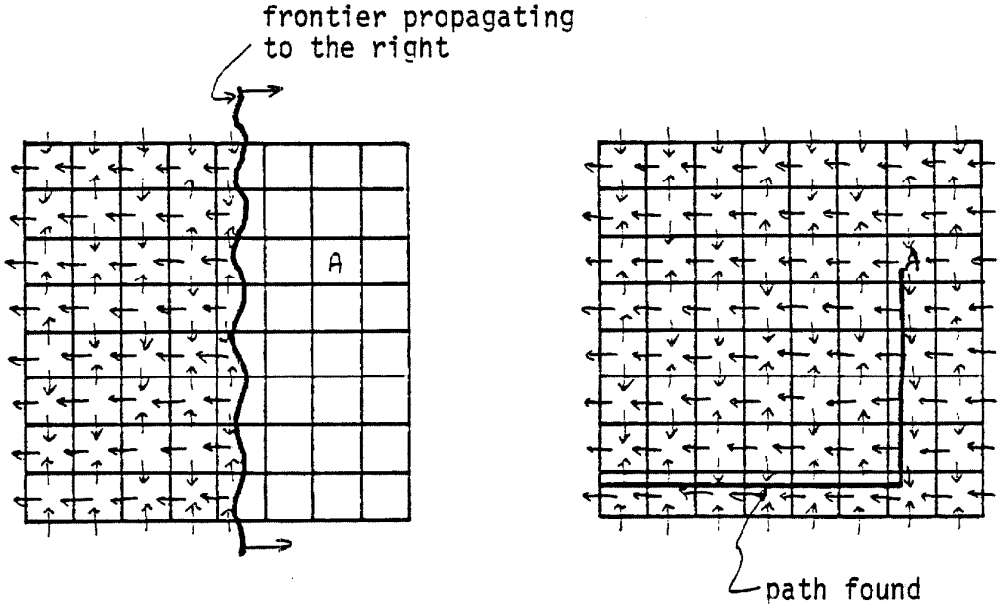


Figure 10-6. Possible result when a plane-wave frontier passes through the cells. Arrows parallel to the frontier erroneously indicate potential vertical paths.

The fix here is to provide more storage to cover the case that the **PATHFINDER** ignored. One way to do this is to move the arrows back into the cells and accept the fact that one of the four possible states of two arrows in adjacent cells will never happen. Another possibiliy is to use a "tri-flop", a circuit with three stable states, as the "arrow" storage between cells. Either solution will remedy this flaw in the **PATHFINDER**.

Despite these three flaws, the **PATHFINDER** chip will successfully find two layer paths between two points. In most cases the effects of these flaws are negligible, and the system performs fully as expected. By making the corrections outlined in this chapter, the **PATHFINDER** can be improved to eliminate the occasional irregularities that occur with the first design.

Chapter 11

LESSONS TAUGHT BY THE PATHFINDER

I have demonstrated one example of a processing system that effectively uses a hybrid of digital and analog information. While such an effort is a significant achievement, one should not merely accept the result as a job well done and move on to the next task. The design of a novel system such as the PATHFINDER chip involves many new ideas, and these new insights can often lead to important breakthroughs in other related areas. Thus, it is important to take note of the lessons learned from such a design experience.

This chapter will identify some important lessons that the PATHFINDER's design taught me. With respect to hybrid processing, in particular, the lessons concern how to represent the data, how to achieve interaction between

the analog and digital data, how to deal with non-ideal environments, and how to interpret the results of the processing. An unexpected result of this research was an important lesson concerning the communication side of processing in general. An exploration of these topics will complete this excursion into the digital/analog hybrid world.

The problem of representing data in a processing machine is an important one. In a hybrid processor, it is a central issue, since the hybrid nature of the system implies that digital variables will represent some data while analog variables will represent other data. One guideline is made clear by the discussions in the early chapters of this thesis comparing analog and digital processing. Any information that must pass through several processing steps MUST use a digital representation if it is to avoid accumulating errors due to inaccuracies and noise. Thus, in any sort of an iterative process, whether the computation is iterated in time, as in a programming loop, or iterated in space, as in an array processor, the data that pass from one computation to the next must be in digital form. Using digital representations of these variables restores the data to clean, noise free signals to be used by the next

computation. Errors introduced by noise in each individual computation cannot accumulate as they inevitably would if analog representations were used. Analog representations of variables are possible only in computations that occur locally or where the accumulation of noise-induced errors is acceptable.

In any non-trivial processing problem the data that eventually generate the desired result are likely to pass through many computations before producing the answer, and thus these data must be represented digitally. In the **PATHFINDER** system, the data representing the path endpoints are specified, and the goal is to determine how to step from cell to cell, incrementing or decrementing the x, y, or z position variables with each step to proceed from one endpoint to another. If this incrementing or decrementing were done on an analog representation of the path data, errors could cause the calculated position to stray away from the actual desired position after proceeding only a few steps from the starting point and the "path" found would lose its meaning. Only a digital representation makes sense here, so that each step from cell to cell is a known, fixed increment in space with no chance of noise accumulating as the number of steps taken increases.

Since the eventual answer-producing data generally must be in digital form, that leaves only the data that control the processing, i.e. the parameters to the process, as candidates for analog representation. This is in keeping with the requirement that analog variables must not take part in the iterative computation, but may provide some parametric data for each individual computation as it occurs. There are many choices available for representing analog data, depending upon the type of processing system under study. Voltage level, chemical concentration, position, and even temperature might be appropriate analog variables under the right circumstances. The choice really depends on two questions. First, how precisely can one represent the analog data using the given quantity, and second, how well can one recover the data thus represented and use it in calculations? Both of these questions relate to how precisely one can measure the physical quantity being considered to represent the data.

Of all the possible physical quantities that are candidates for representing analog data, time is the one over which we have the most control. Time is a monotonically increasing function. It never stops or moves backwards. One can count on it to steadily progress

in one direction. Thus, when an event occurs, we can record that fact and use it in our calculations without fear that the event will "un-occur" due to some momentary reversal of time. This feature makes time a useful vehicle for transferring analog data from place to place.

In VLSI systems, voltage levels come to mind as candidates for analog data representation, but again time proves to be a better choice. Voltage levels are hard to control on an integrated circuit, and can easily succomb to noise from capacitive coupling with clock signals and other transients, and thus would be a poor choice for analog data representation. On the other hand, time is relatively easy to control because of the ease and precision with which capacitors can be designed and matched. Representing analog data by a time interval between two events leads to a very controllable variable that is quite insensitive to the noise present in a VLSI environment.

The **PATHFINDER** design uses intervals of time to represent the analog cost of propagating the information wavefront through each cell in the array. The delay between the initial arrival of the wavefront at the cell and the transmission of the wavefront on to the next set

of neighbors is a continuous function of the cost for travelling through that cell. The computation that uses the analog input, determining how long to delay the wavefront, takes place entirely within the cell, and the output passed on to the next cells is a digital signal representing the advancing wavefront. All the requirements of variable representation mentioned in the preceding paragraphs are met. Thus, the **PATHFINDER** is a workable hybrid processing system.

In order for data processing to take place, the variables, in all their different representations, must interact with each other in some way. If we are to build VLSI hybrid systems in which the analog parameters are represented as intervals of time, we must design our digital circuitry to be dependent on the relative timing of events. Since the time intervals are continuously variable, the digital machine must be asynchronous if it is to make use of the full precision available in the analog representation. Since the time an event happens has no meaning except in relation to the time of some other event, there must be more than a single process in progress so that there will be two events to compare. Thus, one can think of a general machine of this type as consisting of a number of asynchronous state machines.

The timing of each state machine depends on some analog parameter to the problem, and the operation of each state machine depends upon the relative timing of the inputs it receives from other state machines.

The **PATHFINDER** consists of an array of identical asynchronous finite state machines. The operation performed by each state machine is determined by the relative timing of the inputs, i.e. which input arrives first. The output of each state machine is delayed by a time that is a function of the timing of the inputs and the analog cost parameters. The analog variables interact with the digital processing by varying the relative timing of different events. Since the state machine operation depends upon the relative timing, the analog data successfully interact in the proper way.

Any real processing system will be built in some physical medium, which is almost certain to have some non-ideal characteristics. Digital processing techniques are tolerant to some degreee of irregularities in their environments, but analog techniques often depend on having an ideal, uniform medium as a foundation, and thus a designer of hybrid processors must watch out for and correct any non-idealities present in his design. The

PATHFINDER design includes corrections for two important non-ideal characteristics that the physics of the situation imposes. First, variations in transistor characteristics from chip to chip mean that a different voltage must be applied to the cost setting transistor gates at the bottom of the discharge paths for each different chip. I overcame this non-ideality by using the current mirror arrangement discussed earlier, which allows one to set the discharge currents without worrying about what voltage ends up on those transistor gates. The second serious non-ideality faced in the **PATHFINDER** design is the discontinuity existing at chip boundaries in the array of cells, discussed in Chapter 9. In this case, the fix was the clock signal, which reduced the dependence of the cell to cell delay on the actual delay occurring in the wires. This kind of problem demands individual attention in every system. No real processing system can ever be completely ideal in every way.

Finally, one must be careful in interpreting the results of a hybrid processor. Since the output of such a system is generally digital, as is the path found by **PATHFINDER**, one tends to judge the output by digital standards. This is an unfair comparison, since part of the computation in a hybrid system is analog processing

and as such is subject to the limited accuracy that is characteristic of analog techniques. Thus, one should expect that hybrid systems will occasionally produce results that are not strictly correct by digital standards.

The **PATHFINDER** uses analog techniques to set the cost for travelling through each cell. It achieves this goal by varying the delay in advancing the wavefront during propagation. Since this delay is set by analog techniques, it cannot be specified with absolute accuracy. Variations in capacitance and transistor thresholds from cell to cell will inevitably cause some cells to pass the wavefront through a little more quickly than intended, while others pass it on a little more slowly. The consequence is that during propagation, the symmetrically expanding wavefront may develop some small bends or perturbations in its shape. It is thus possible that the **PATHFINDER** system will select a path that is slightly more costly than minimal. However, if the inaccuracies in delays are randomly distributed through the array, the unwanted "errors" will be integrated out as the wavefront sweeps through the cells. In any event, the hybrid processing system is in no danger of losing its signal "in the noise" as a purely analog system would.

The **PATHFINDER** provides an unexpected lesson about the communication side of processing. In this system, the communication scheme consists of passing signals from cell to cell to indicate wavefront expansion during the propagation process. The signals are delayed as they pass through the cells to implement the required cost functions. Recall that in Chapter 10, one of the flaws mentioned involved the placement of the delay elements. Positioning them within the cell as is done in the **PATHFINDER** cannot correctly implement the cost system as intended. Only by placing the delays on the input or output of the cells, that is, on the communication paths themselves, can the problem be correctly solved. This result demonstrates the importance of the communication side of processing.

Seen from the point of view of communication, the **PATHFINDER** measures distance between cells by the time needed to communicate data between them. With the rectangular array of cells and with a uniform communication cost imposed, this results in the typical case where each cell has four nearest neighbors. By increasing the horizontal communication cost relative to the vertical communication cost, the **PATHFINDER** modifies the effective topology so that now the neighbors in the

vertical direction are "nearer" than the neighbors in the horizontal direction. This suggests that more complicated control over communication costs might allow the implementation of more complicated effective topologies in the plane of the silicon chip. One can demonstrate this effect by observing how communication works for other more familiar situations.

Consider the various communities in the Los Angeles area, and the network of roads that interconnect them. Each community has its own set of roads for local communication. Communication, or travel between adjacent communities is easy, using the roads that cross between the two areas. Communication between non-adjacent communities would be difficult in comparison, if it were not for the freeway system. The freeway system provides a low cost communication system linking the communities. The existence of the freeways makes possible the interaction of distant communities that would otherwise have been too costly. The freeways change the effective topology of the area from a two-dimensional plane in which interaction can occur only between neighbors to a more complex structure allowing cheap interaction with a more varied population.

One can find a second example along the same line in the human nervous system. The interconnections, and thus the communication patterns, in the brain are only slightly understood. Nevertheless, one could think of measuring the cost of neural processing in terms of the time it takes to propagate the nerve impulses through the required neurons. Interestingly, although most neurons transmit impulses at about the same speed, there is a special class of neurons that are sheathed with a material that allows much faster impulse propagation [12]. One can only guess that these less costly communication paths exist to enrich the topology of the interconnections in the brain, just as the freeways do in Los Angeles, to allow more extensive communication.

The two examples above suggest an exciting possible extension to integrated circuit design. In each case, the majority of the communication happens locally, using the ordinary communication paths, whether built from concrete or protoplasm. However, a second level of interconnections exists to provide low cost long distance communication. These second-level paths are more expensive or complex, just as freeways are more expensive to build than ordinary roads, but only a relatively small number of them are needed to provide adaquate coupling

between distant areas. This suggests that integrated circuit design could benefit from an extra, high speed communication layer, even if the complexity of that layer were very limited by fabrication or design difficulties. The ability of such a low-cost communication path to de-planarize the surface of the silicon could lead to easier implementation of some processing problems.

The conclusion of this thesis is that hybrid processing can play an important role in the design of today's computing machines. The success of the **PATHFINDER** chip in solving the two-layer path finding problem is evidence that designers should include analog techniques in their bag of tricks in order to realize the full potential of computing power available to them. By applying the lessons learned in the design of the **PATHFINDER**, hybrid processing is a viable approach in any situation when digital decisions are made based on analog data.

In the early 1960's, this conclusion may have evolved more easily. At that time, the digital revolution was only just under way. Designers were not as brainwashed into digital thinking as they are today. One author[3]

seems to have realized the potential of hybrid processing when he wrote,

> "Once austerely separate, analog and digital techniques are beginning to intermingle freely and complement each other. Computers of the future undoubtedly will borrow freely from both digital and analog techniques and may eventually become true 'hybrids,' merging the characteristics of both types so completely that they are no longer separately identifiable."

With the design of the **PATHFINDER**, hybrid processing has arrived. Designers of today's processing equipment demand high performance from their circuits. Only by using hybrid processing techniques can they have access to the full power of the physical structure upon which they base their circuits. It is time now to recognize hybrid processing as an alternative to other types of design and to begin applying it to important computing problems.

REFERENCES

1. Akers, S., "A Modification of Lee's Path Connection Algorithm," IEEE Trans. Electronic Computers (Short Notes), Vol. EC-16, pp. 97-98, February, 1967.

2. Cheng, Edmund, "A Single-Chip Cursive Character Generator," Ph.D. Thesis, California Institute of Technology, 1976.

3. Jocobowitz, Henry, Electronic Computers, Doubleday and Company, Inc., Cedar City, New York, 1963.

4. Kohavi, Zvi, Switching and Finite Automata Theory, McGraw Hill Book Company, New York, 1971.

5. Lee, C., "An Algorithm for Path Connections and its Applications," IEEE Trans. Electronic Computers, Vol. EC-10, pp. 346-365, September, 1961.

6. Mead, Carver and Lynn Conway, Introduction to VLSI Systems, Addison-Wesley Publishing Company, Reading, Mass. 1980.

7. Moore, E., "Shortest Path Through a Maze," Annals of the Computation Laboratory of Harvard University, Vol. 30, Cambridge, Mass.: Harvard University Press, pp. 285-292, 1959.

8. Newman, William, and Robert Sproull, Principles of Interactive Computer Graphics, McGraw Hill Book Company, New York, 1973.

9. Posna, John, "Multi-valued Logic Takes New Paths," Electronics, pp. 100-103, February 24, 1981.

10. Posna, John, "Four-State Cell Doubles ROM Bit Capacity," Electronics, p. 39, October 9, 1980.

11. Rattner, Justin, and William Lattin, "Ada Determines Architecture of 32-bit Microprocessor," Electronics, pp. 119-126, February 24, 1981.

12. Stephens, Charles, "The Neuron," Scientific American, pp. 55-65, September, 1979.